МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ
"КИЕВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ
ИМЕНИ ИГОРЯ СИКОРСКОГО"
КАФЕДРА ЗВУКОТЕХНИКИ И РЕГИСТРАЦИИ ИНФОРМАЦИИ

ОСНОВЫ ИСПОЛЬЗОВАНИЯ SQL В СЕРВЕРНЫХ СИСТЕМАХ

Учебное пособие

(для студентов-иностранцев по специальности 171 “Электроника” и направлению подготовки ”6.050903 “Телекоммуникации”)

2016

УДК 004.738.5
ББК 32.988-02-018
В16

Основы использования SQL в серверных системах: Учебное пособие для студентов-иностранцев по специальности 171 “Электроника” и направлению подготовки 6.050903 “Телекоммуникации” / А.Г. Власюк, Г.Н. Розоринов, Н.И. Чичикало, К.А. Трапезон. – К.: Аверс, 2016. – 125 с.

Гриф “Затверджено Вченою радо КПІ ім. Ігоря Сікорського” надано за поданням Методичної ради університету на засіданні Вченої ради Національного технічного університету України “Київський політехнічний інститут імені Ігоря Сікорського” (протокол №4 від 3 квітня 2017 року)

Составители:

Власюк Анна Григорьевна, доктор техн. наук, профессор.

Розоринов Георгий Николаевич, доктор техн. наук, профессор.

Чичикало Нина Ивановна, доктор техн. наук, профессор.

Трапезон Кирилл Александрович, кандидат техн. наук, доцент

Ответственный редактор проф. Савченко Ю.Г.

Рецензенты: Петрищев О.М., профессор, д.т.н.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Ни одна система автоматизированного проектирования электронных устройств (серверов) не может обойтись без баз данных (БД). Изучение целесообразно проводить на основе базы данных Oracle9i (или версии 11), так как корпорация Oracle является общепризнанным лидером по созданию баз данных. Доказательством этого может служить то, что почти все надстройки, разрабатываемые Oracle над языком Structured Query Language (SQL), включаются в последующую разрабатываемую редакцию стандарта ANSI SQL. Язык SQL, служит для удобного и понятного пользователям формулирования обращений к реляционным БД и манипулирования данными. Язык обеспечивает общее управление данными в приложениях, где требуется гибкость структур данных и маршрутов доступа, специфическое манипулирование данными и изменение степени ограничения доступа. Язык SQL, в основном, является языком запросов и манипулирования данными, однако его функции и возможности значительно шире. Он обеспечивает построение эффективных диалоговых систем обработки транзакций со стандартизированными наборами параметризируемых запросов. В языке имеются следующие средства: определения и манипулирования схемой БД; определения ограничений и контроля целостности; определения и уничтожения структур физического уровня для эффективной реализации транзакций; авторизации и защиты доступа к данным; использования точек сохранения транзакций и откатов при сбоях; средства динамической компиляции запросов, на базе которых возможна диалоговая их обработка.

Целью данного учебного пособия является изучение основных команд SQL – языка доступа к базам данных. Данный язык является стандартом для всех реляционных баз данных. Реляционная база данных – это база данных, представляемая в виде совокупности таблиц, и удовлетворяющая правилам Кодда, основным из которых является целостность базы данных [1]. Гибкость настройки, необходимость хранения и анализа базы данных привели к тому, что БД используются практически повсеместно, начиная от

Интернет сайтов, малых предприятий, банков до операторов мобильной связи.

Системы управления базами данных (СУБД) – это инструмент доступа к базам данных. В зависимости от задач, обрабатываемых СУБД, к ней ставятся соответствующие требования. Т.е. в одних задачах критичным является время записи в базу данных информации (например, система фиксации звонков оператора мобильной связи), в других же наоборот время записи данных не критично, но должна быть высокая скорость чтения накопленной информации (например, системы бухгалтерской и других отчетностей). В связи с этим различают три основных типа СУБД: OLTP (on-line transaction processing) - оперативная обработка транзакций (в реальном времени); OLAP (on-line analytical processing) - аналитическая обработка в реальном времени (технология обработки информации, включающая составление и динамическую публикацию отчетов и документов); DSS (Decision Support System) - система поддержки принятия решений.

Обычно СУБД, например, такие как Oracle8i-10g, поддерживают все перечисленные выше типы работы. Быстродействие в той или иной области зависит только от настроек СУБД. Поэтому на стадии проектирования системы нужно решить задачу оптимизации: быстродействие при записи данных или при их считывании, и сделать соответствующие настройки, либо найти компромиссное решение, которое, обычно, стараются реализовать в системах DSS. Такими настройками занимается администратор базы данных (DBA), и они не будут рассматриваться в этом пособии.

В качестве базы данных для хранения информации об объекте была выбрана наиболее популярная и распространенная на сегодня база данных Oracle 9i, поскольку – это надежная, хорошо зарекомендовавшая себя база данных, имеющая несколько средств защиты данных от потери (undo сегмент, архивация логов, проводимых по базе операций изменения данных, и мощный инструмент создания резервных копий и восстановления поврежденных баз – RMAN).

ЗАПУСК СЕРВЕРА ORACLE9I

Студенты работают за клиентскими компьютерами, и подключаются к серверу базы данных с помощью iSQL*Plus, работающего через Интернет браузер.

Запуск сервера производится преподавателем. Для запуска необходимо выполнить следующие шаги:

- В «Службах» Microsoft Windows (Пуск -> Панель управления -> Администрирование -> Службы) запустить службы OracleTNSListener и OracleSID.
- Открыть командную строку (cmd), подключиться к инстанции базы данных командой sqlplus “/as sysdba”
- В sql*plus выполнить команду startup. Дождаться завершения исполнения команды. После этих шагов база данных запущена, и готова к работе.
- Запустить Web-сервер (Пуск -> Oracle9i -> Apache server -> Start HTTP server), для того, что бы студенты могли подключиться к базе данных, используя iSQL*Plus.

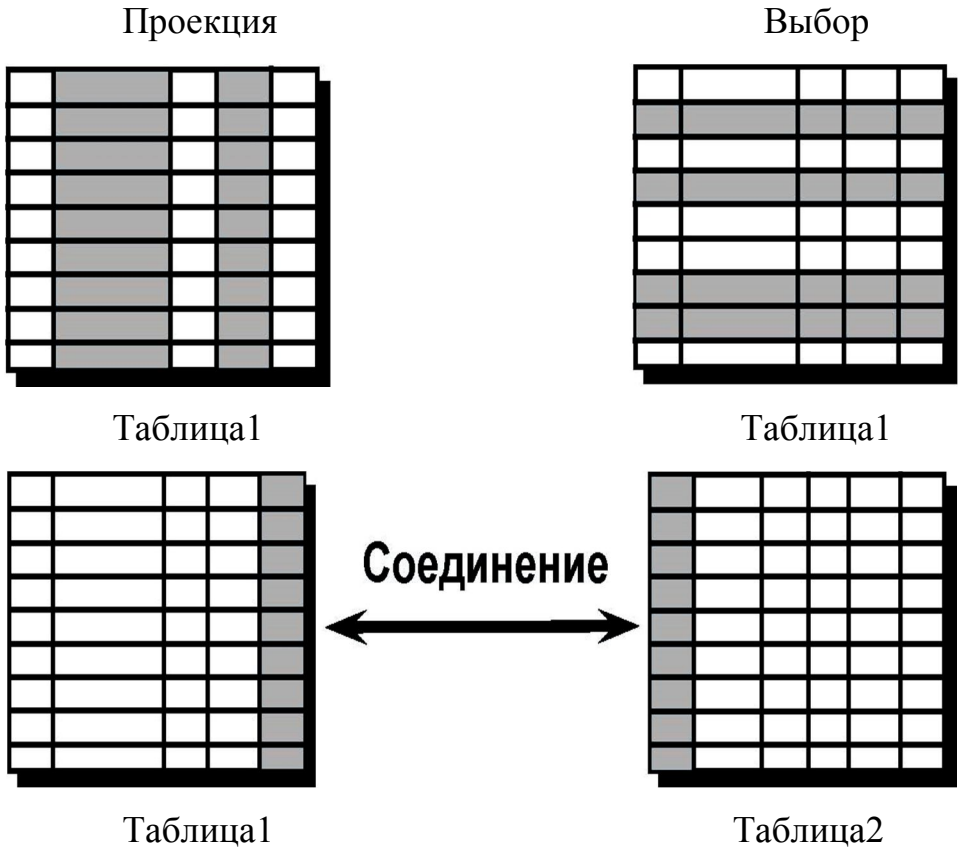
ТЕМА 1. БАЗОВЫЕ КОМАНДЫ ВЫБОРА ЯЗЫКА SQL

Рассматриваемые вопросы:

- Выборка данных из разных таблиц;
- Описание структуры таблиц;
- Выполнение арифметических вычислений и задание имен столбцов;
- Использование iSQL*Plus;
- Возможности команды **SELECT** языка SQL;
- Выполнение базовой команды SELECT;
- Различия между командами SQL и iSQL*Plus.

Запрос это команда, которая даётся вашей программе базы данных и которая сообщает ей, что нужно вывести определённую информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран компьютера или терминала, хотя в большинстве случаев её можно также послать на принтер, сохранить в файле (как объект в памяти компьютера) или предоставить как вводную информацию для другой команды или процесса. Запрос не меняет информацию в таблицах, а просто показывает её пользователю. Любой запрос SQL имеет в своём составе одну команду. Структура этой команды обманчиво проста, потому что вы можете расширять её так, чтобы выполнить сложные оценки и обработку данных. Эта команда называется **SELECT** (ВЫБРАТЬ).

Возможности команды SELECT языка SQL



В самой простой форме команда **SELECT** просто инструктирует БД, чтобы извлечь информацию из таблицы.

Базовая команда SELECT

```
SELECT  *|{[DISTINCT] столбец|выражение [псевдоним],...}
FROM    таблица;
```

- **SELECT** указывает, какие столбцы;
- **FROM** указывает, из какой таблицы.

FROM - ключевое слово, подобное **SELECT**, которое должно быть представлено в каждом запросе. Оно сопровождается пробелом и именем таблицы, используемой в качестве источника информации.

Точка с запятой используется во всех интерактивных командах SQL, чтобы сообщать базе данных, что команда записана и готова к выполнению. В некоторых системах индикатором конца команды является обратный слэш (\) в строке.

Другими словами, эта команда просто выводит все данные из таблицы. Большинство программ будут также давать заголовки столбца, как выше, а некоторые позволяют определить детальное форматирование вывода, но это уже вне стандартной спецификации.

Выбор всех столбцов

Если необходимо видеть все столбцы таблицы, имеется необязательное сокращение, которое вы можете использовать – “звездочка (*)”. Звёздочка (*) может применяться для вывода полного списка столбцов следующим образом

```
SELECT *
FROM   departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

8 rows selected.

«*» заменяет перечисление всех столбцов, используется порядок по умолчанию.

Выбор конкретных столбцов

Команда **SELECT** способна извлечь строго определенную информацию из таблицы. Сначала мы можем предоставить возможность

увидеть только определённые столбцы таблицы. Это выполняется легко: простым исключением столбцов, которые вы не хотите видеть, из команды **SELECT**.

```
SELECT department_id, location_id
FROM   departments;
```

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

Если необходимо выбрать не все столбцы таблицы, а только определенные или нужно изменить порядок вывода столбцов, то необходимые столбцы перечисляются в поле **SELECT** через запятую.

Написание команд SQL:

- Команды SQL не различают регистры символов;
- Команды SQL могут занимать одну или несколько строк;
- Ключевые слова нельзя сокращать и размещать на двух строках;
- Предложения обычно пишутся на отдельных строках;
- Для облегчения чтения используются табуляция и отступы.

Заголовки столбцов по умолчанию

- iSQL*Plus
  - Выравнивание по умолчанию: по центру
  - Вывод по умолчанию: в символах верхнего регистра
- SQL*Plus
  - Слева: даты и символьные данные

– Справа: цифровые данные
– Вывод по умолчанию: в символах верхнего регистра

Арифметические выражения

Создаются из данных типа **NUMBER** и **DATE** с помощью арифметических операторов:

Оператор	Описание
+	Сложение
-	Вычитание
*	Умножение
/	Деление

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

LAST_NAME	SALARY	SALARY+300
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300
Hunold	9000	9300
Ernst	6000	6300
Lorentz	4200	4500
Gietz	8300	8600

20 rows selected.

Приоритеты операторов

- Умножение и деление имеют приоритет над сложением и вычитанием;
- Операторы с одинаковым приоритетом выполняются с лева на право;
- Для выполнения операторов в определенном порядке и упрощения их чтения используются скобки.

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

LAST_NAME	SALARY	12*SALARY+100
King	24000	288100
Kochhar	17000	204100
De Haan	17000	204100
Hunold	9000	108100
Ernst	6000	72100
Lorentz	4200	50500
Gietz	8300	99700

20 rows selected.

Использование скобок

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

LAST_NAME	SALARY	12*(SALARY+100)
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200
Hunold	9000	109200
Ernst	6000	73200
Lorentz	4200	51600
Gietz	8300	100800

20 rows selected.

Неопределенное значение (NULL)

Часто в таблице будут записи, которые не имеют никаких значений поля, например, потому что информация не завершена, или потому что это поле просто не заполнялось. SQL учитывает такой вариант, позволяя вам вводить значение **NULL** (ПУСТОЙ) в поле, вместо значения. Когда значение поля равно NULL, это означает, что программа базы данных специально промаркировала это поле как не имеющее никакого значения для этой строки (записи).

Это отличается от просто назначения полю значения нуль или пробела, которые база данных будет обрабатывать так же, как и любое другое значение. Точно так же как **NULL** не является техническим значением, оно не имеет и типа данных. Оно может помещаться в любой тип поля. Тем не менее, **NULL** в SQL часто упоминается как "нуль".

- Неопределенное значение (**NULL**) – это значение, которое недоступно, не присвоено, неизвестно или неприменимо.
- Это не ноль и не пробел.

```
SELECT last_name, job_id, salary, commission_pct
FROM   employees;
```

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
King	AD_PRES	24000	
Kochhar	AD_VP	17000	
Zlotkey	SA_MAN	10500	.2
Abel	SA_REP	11000	.3
Taylor	SA_REP	8600	.2
Higgins	AC_MGR	12008	
Gietz	AC_ACCOUNT	8300	

20 rows selected.

Неопределенные значения в арифметических выражениях

Результат вычисления выражения, содержащего неопределенное значение, также будет неопределенным.

```
SELECT last_name, 12*salary*commission_pct
FROM   employees;
```

LAST_NAME	12*SALARY*COMMISSION_PCT
King	
Kochhar	
Zlotkey	25200
Abel	39600
Taylor	20640
Higgins	
Gietz	

20 rows selected.

Псевдоним (алиас) столбца

- Альтернативный заголовок столбца.

- Удобен при вычислениях.
- Следует сразу за именем столбца; ключевое слово AS между именем столбца и псевдонимом необязательно.
- Заключается в двойные кавычки, если содержит пробелы, специальные символы или различает регистры символов.

Использование псевдонимов столбцов

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

NAME	COMM
King	
Kochhar	
Higgins	
Gietz	

20 rows selected.

```
SELECT last_name  "Name",
       salary*12 "Annual Salary"
FROM   employees;
```

Name	Annual Salary
King	288000
Kochhar	204000
Higgins	144000
Gietz	99600

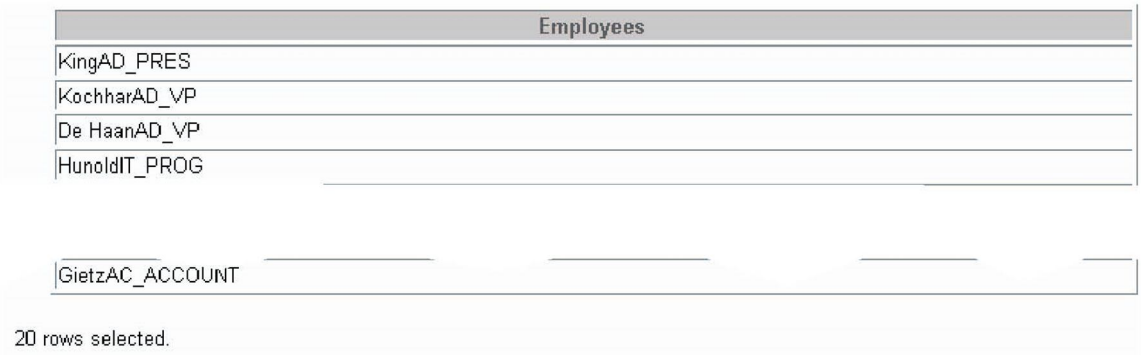
20 rows selected.

Оператор конкатенации

- Соединяет столбцы или символьные строки с другими столбцами.
- Изображается двумя вертикальными линиями (||).
- Создает столбец с результатом, представляющим символьное выражение.

Использование оператора конкатенации

```
SELECT   last_name||job_id AS "Employees"
FROM     employees;
```



Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP
HunoldIT_PROG
GietzAC_ACCOUNT

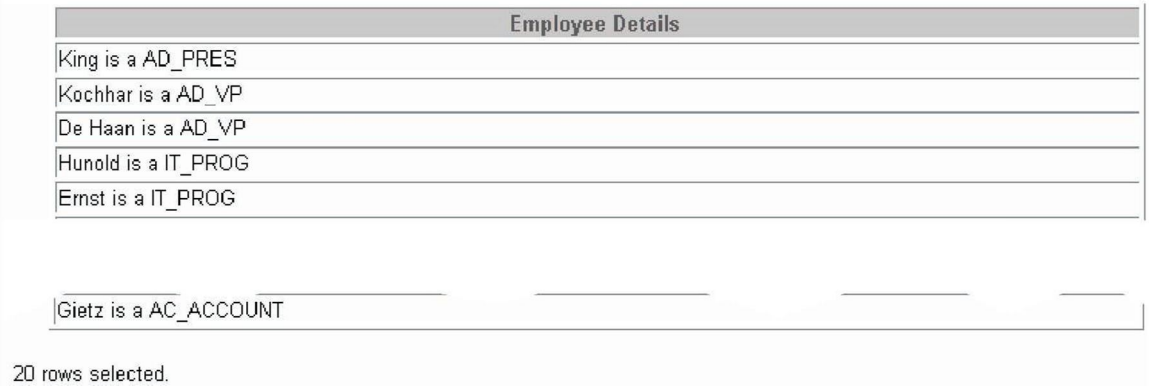
20 rows selected.

Строка символов – литерал

- Литерал – это символ, число или дата, включенные в **SELECT** список.
- Даты и символьные литералы должны быть заключены в апострофы.
- Каждая символьная строка выводится один раз для каждой возвращаемой строки таблицы.

Использование символьных литералов

```
SELECT last_name  ||' is a '||job_id
       AS "Employee Details"
FROM   employees;
```



Employee Details
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Gietz is a AC_ACCOUNT

20 rows selected.

Дублирование строк

По умолчанию выдаются все строки, включая дубликаты.

```
SELECT department_id
FROM   employees;
```

DEPARTMENT_ID
90
90
90
60
60
60
50
50

20 rows selected.

Устранение строк-дубликатов

Дубликаты устраняются с помощью ключевого слова **DISTINCT** в команде **SELECT**. **DISTINCT** (ОТЛИЧИЕ) - аргумент, который обеспечивает способ устранения дублирующих значений из предложения SELECT. Другими словами, **DISTINCT** следит за тем, какие значения были ранее, чтобы они не дублировались в списке. **DISTINCT** может указываться только один раз в данном предложении **SELECT**.

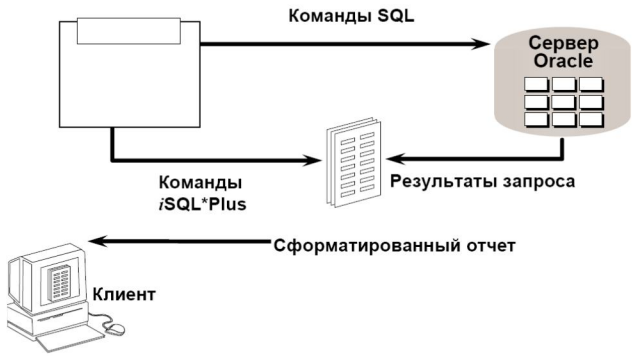
Вместо **DISTINCT** вы можете указать **ALL**. Это будет иметь противоположный эффект, дублирование строк вывода сохранится. Так как это - тот самый случай, когда вы не указываете ни **DISTINCT** ни **ALL**, то **ALL** - по существу скорее пояснительный, а не действующий аргумент.

```
SELECT DISTINCT department_id
FROM   employees;
```

DEPARTMENT_ID
10
20
50
60
80
90
110

8 rows selected.

Взаимодействие SQL и iSQL*Plus



Сравнение команд SQL и iSQL*Plus

SQL	iSQL*Plus
• Язык • Стандарт ANSI • Сокращать ключевые слова нельзя • Команды манипулируют данными и определениями таблиц в базе данных	• Среда • Разработка Oracle • Сокращать ключевые слова можно • Команды не позволяют манипулировать данными в базе данных • Выполняется в браузере • Загружается централизованно, не должен быть размещен на каждой машине

Обзор iSQL*Plus

После входа в iSQL*Plus можно:

- Получить описания структуры таблиц;
- Отредактировать команды SQL;
- Выполнить команды SQL из iSQL*Plus;
- Сохранить команды SQL в файлах и добавить команды SQL к файлам;
- Выполнить сохраненные файлы;
- Загрузить команды из файла в окно редактирования iSQL*Plus.

Вывод структуры таблицы

Вывод структуры таблицы производится с помощью команды DESCRIBE iSQL*Plus

```
DESC[RIBE] имя_таблицы
```

Пример:

```
DESCRIBE employees
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Выводы по теме №1

В этой теме вы познакомились с тем, как задавать команду **SELECT**, по которой:

- возвращаются все строки и столбцы таблицы;

- возвращаются определенные столбцы таблицы;
- используются псевдонимы для изменения наименований заголовков столбцов:

```
SELECT    *|{[DISTINCT] столбец|выражение [псевдоним],...}
FROM      таблица;
```

Контрольные задания по теме №1

1. Инициируйте сессию iSQL*Plus, используя предоставленный логин и пароль.
2. Команды iSQL*Plus – это команды доступа к базе данных? Да/Нет.
3. Будет ли успешно выполнен следующий запрос?

```
SELECT last_name, job_id, salary AS Sal
FROM employees;
```

Да/Нет.

4. Будет ли успешно выполнен следующий запрос?

```
SELECT *
FROM job_grades;
```

Да/Нет.

5. В предоставленном запросе есть четыре ошибки. Вы можете их выявить?

```
SELECT employee_id, last_name
sal x 12 ANNUAL SALARY
FROM employees;
```

6. Отобразите структуру таблицы DEPARTMENTS. Запросите из неё все данные.
7. Покажите структуру таблицы EMPLOYEES. Создайте запрос для отображения фамилии, job_id, даты приема на работу и employee_id, для всех сотрудников, так что бы employee_id был первым столбцом в результате. Сохраните запрос в файле lab1_7.sql.
8. Запустите запрос из файла lab1_7.sql.

9. Создайте запрос для отображения уникальных типов работ (job_id) из таблицы EMPLOYEES.

10. Откройте запрос из файла lab1_7.sql. Дайте столбцам заголовки Emp#, Employee, Job и Hire Date, соответственно. Запустите запрос.

11. Отобразите фамилию, объединенную с типом работы, разделяя их запятой и пробелом. Назовите столбец Employee and Title.

12. Создайте запрос для отображения всех данных из таблицы EMPLOYEES. Разделяйте столбцы запятыми. Назовите получившийся столбец THE_OUTPUT.

ТЕМА 2. ОГРАНИЧЕНИЕ И СОРТИРОВКА ВЫХОДНЫХ ДАННЫХ

Рассматриваемые вопросы:

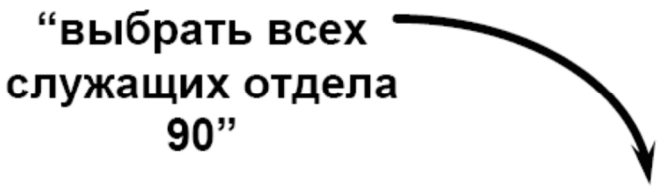
- Ограничение количества строк, возвращаемых запросом.
- Сортировка возвращаемых строк.
- Выборка данных и изменение последовательности вывода строк.
- Ограничение количества возвращаемых строк с помощью предложения **WHERE**.
- Сортировка строк с помощью предложения **ORDER BY**.

Ограничение количества выбираемых строк путем отбора

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90
103	Hunold	IT_PROG	60
104	Ernst	IT_PROG	60
107	Lorentz	IT_PROG	60
124	Mourgos	ST_MAN	50

20 rows selected.



EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Количество возвращаемых строк можно ограничить с помощью предложения **WHERE**. **WHERE** - предложение команды **SELECT**, которое позволяет устанавливать предикаты, условие которых может быть или верным (true), или неверным (false) для любой строки таблицы. Команда извлекает только те строки из таблицы, для которых такое утверждение

верно. Когда предложение **WHERE** предоставлено, программа базы данных просматривает всю таблицу построчно и исследует каждую строку, чтобы определить, верно ли утверждение.

Предложение **WHERE** **<u>всегда</u>** следует за предложением **FROM**.

```
SELECT    *|{[DISTINCT] столбец|выражение [псевдоним],...}
FROM      таблица
[WHERE    условие(я)];
```

Использование предложения WHERE

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Символьные строки и даты

Символьные строки и даты заключаются в апострофы.

В символьных значениях различаются регистры символов, а в датах – форматы.

Формат дат по умолчанию – DD-MM-RR (число-месяц-год).

```
SELECT last_name, job_id, department_id
FROM   employees
WHERE  last_name = 'Goyal';
```

Операторы сравнения

Оператор	Значение
=	Равно
>	Больше, чем
>=	Больше или равно
<	Меньше, чем
<=	Меньше или равно
<>	Не равно

Эти операции имеют стандартное значение для чисел. Для символов их определение зависит от формата преобразования, ASCII или EBCDIC. SQL сравнивает символьные значения в терминах основных чисел, как определено в формате преобразования. Даже значение символа, такого как "1", который представляет число, не обязательно равняется числу, которое он представляет. Вы можете использовать реляционные операции, чтобы установить алфавитный порядок, например, "a" < "n", где а идёт раньше в алфавитном порядке, но всё это ограничивается с помощью параметра преобразования формата.

И в ASCII, и в EBCDIC символы сортируются по значению: символ имеет значение меньше, чем все другие символы, которым он предшествует в алфавитном порядке и которые имеют с ним один вариант регистра (верхний или нижний). В ASCII все символы верхнего регистра меньше, чем все символы нижнего регистра, поэтому "Z" < "a", а все числа - меньше чем все символы, поэтому "1" < "Z". То же относится и к EBCDIC.

Использование условий сравнения

```
SELECT last_name, salary
FROM   employees
WHERE  salary <= 3000;
```

LAST_NAME	SALARY
Matos	2600
Vargas	2500

В дополнение к реляционным и булевым операциям SQL использует специальные операторы: **IN, BETWEEN, LIKE и IS NULL**.

Другие условия сравнения

Оператор	Значение
BETWEEN ...AND...	Находится в диапазоне от одного значения до другого (включительно)
IN(список)	Совпадает с каким-либо значением списка
LIKE	Соответствует символьному шаблону
IS NULL	Является неопределенным значением

Использование условия IN

Оператор **IN** определяет набор значений, в который данное значение может или может не быть включено.

Условие принадлежности **IN** используется для проверки на вхождение значений в список.

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201);
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

8 rows selected.

Оператор **IN** определяет набор значений с помощью имён членов набора, заключённых в круглые скобки и разделённых запятыми. Он затем проверяет различные значения указанного поля, пытаясь найти совпадение со значениями из набора. Если это случается, то предикат верен. Когда набор содержит числовые значения, а не символы, одиночные кавычки опускаются.

Использование условия BETWEEN

Оператор **BETWEEN** похож на оператор **IN**. Но, в отличие от определения по числам из набора, как это делает **IN, BETWEEN** определяет диапазон, значения которого должны уменьшаться, что делает предикат верным. Вы должны ввести ключевое слово **BETWEEN** с начальным значением, ключевое **AND** и конечное значение. В отличие от **IN, BETWEEN** чувствителен к порядку, и первое значение в предложении должно быть последним по алфавитному или числовому порядку.

Условие **BETWEEN** используется для вывода строк на основе диапазона значений

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN  2500 AND 3500;
```

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

SQL не делает непосредственной поддержки невключения граничных значений **BETWEEN. BETWEEN** может работать с символьными полями в терминах эквивалентов ASCII. Это означает, что вы можете использовать **BETWEEN** для выборки ряда значений из упорядоченных по алфавиту значений.

Использование условия LIKE

LIKE применим только к полям типа CHAR или VARCHAR, с которыми он используется для поиска подстрок. Т.е. он ищет поле символа, чтобы увидеть, совпадает ли с условием часть его строки. В качестве условия он использует групповые символы-шаблоны (wildcards) - специальные символы, которые могут соответствовать чему-нибудь.

Имеются два типа шаблонов, используемых с **LIKE**:

- символ подчёркивания (_) замещает любой одиночный символ. Например, 'b_t' будет соответствовать словам 'bat' или 'bit', но не будет соответствовать 'brat'.

- знак процента (%) замещает последовательность любого количества символов (включая символы нуля). Например '%p%t' будет соответствовать словам 'put', 'posit', или 'opt, но не 'spite'.

Условие **LIKE** используется для поиска символьных значений по шаблону с метасимволами.

То есть, условия поиска могут включать алфавитные и цифровые символы:

- % обозначает ни одного или много символов;
- _ обозначает один символ.

Для поиска "%" или "_" можно использовать идентификатор **ESCAPE** .

```
SELECT   first_name
FROM     employees
WHERE    first_name LIKE 'S%';
```

Метасимволы можно комбинировать.

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%';
```

LAST_NAME
Kochhar
Lorentz
Mourgos

LIKE может быть удобен, если вы ищете имя или другое значение и если вы не помните, как они точно пишутся.

Использование условия NULL

Так как **NULL** указывает на отсутствие значения, вы не можете знать, каков будет результат любого сравнения с использованием **NULL**. Когда **NULL** сравнивается с любым значением, даже с другим таким же **NULL**, результат будет ни true, ни false, он неизвестен/undefined. Неизвестный булев вообще ведёт себя так же, как неверная строка, которая, произведя неизвестное значение в предикате, не будет выбрана запросом.

Имейте в виду, что, в то время как **NOT** (неверное) равняется верно, NOT (неизвестное) равняется неизвестно. Следовательно, выражение типа 'city = NULL' или 'city IN (NULL)' будет неизвестно в любом случае.

Часто вы должны отличать неверно и неизвестно между строками, содержащими значения столбцов, которые не соответствуют условию

предиката и которые содержат **NULL** в столбцах. По этой причине SQL предоставляет специальный оператор **IS**, который используется с ключевым словом **NULL**, для размещения значения **NULL**.

С помощью оператора **IS NULL** производится проверка на неопределенные значения.

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL;
```

LAST_NAME	MANAGER_ID
King	

Логические условия

Основные булевы операции также распознаются в SQL. Выражения Буля являются или верными/true, или неверными/false, подобно предикатам. Булевы операции связывают одно или более верных/неверных значений и производят единственное верное или неверное значение.

Стандартными булевыми операциями, распознаваемыми в SQL, являются **AND**, **OR** и **NOT**. Существуют другие, более сложные булевы операции (типа "исключающее ИЛИ"), но они могут быть сформированы из этих трёх простых операций - **AND**, **OR**, **NOT**.

Оператор	Значение
AND	Возвращает результат ИСТИННО, если выполняются *оба* условия.
OR	Возвращает результат ИСТИННО, если выполняется *любое из* условий.
NOT	Возвращает результат ИСТИННО, если следующее условие не выполняется.

Использование оператора AND

Оператор **AND** (“И“) требует выполнения обоих условий.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >=10000
AND    job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

Использование оператора OR

Оператор **OR** (“ИЛИ“) требует выполнения любого из условий.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR     job_id LIKE '%MAN%';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

8 rows selected.

Использование оператора NOT

Обратите внимание, что операция NOT должна предшествовать булевой операции, чьё значение должно измениться, и не должна помещаться перед реляционной операцией.

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

10 rows selected.

Приоритеты операторов

Порядок вычисления	Оператор
1	Арифметические операторы
2	Оператор конкатенации
3	Операторы сравнения
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	NOT
7	AND
8	OR

Изменить стандартную последовательность вычислений можно с помощью круглых скобок.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = 'SA_REP'
OR     job_id = 'AD_PRES'
AND    salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Abel	SA_REP	11000
Taylor	SA_REP	8600
Grant	SA_REP	7000

Использование скобок для изменения порядка вычислений.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  (job_id = 'SA_REP'
OR     job_id = 'AD_PRES')
AND    salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

Предложение ORDER BY

Предложение **ORDER BY** используется для сортировки строк

– **ASC**: сортировка по возрастанию (используется по умолчанию)

– **DESC**: сортировка по убыванию

В команде **SELECT** предложение **ORDER BY** **<u>всегда</u>** указывается последним.

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91
[illegible]	[illegible]	90	13-[illegible]

20 rows selected.

Сортировка в порядке убывания

```
SELECT   last_name, job_id, department_id, hire_date
FROM     employees
ORDER BY hire_date DESC;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Zlotkey	SA_MAN	80	29-JAN-00
Mourgos	ST_MAN	50	16-NOV-99
Grant	SA_REP		24-MAY-99
Lorentz	IT_PROG	60	07-FEB-99
Vargas	ST_CLERK	50	09-JUL-98
Taylor	SA_REP	80	24-MAR-98
Matos	ST_CLERK	50	15-MAR-98
Fay	MK_REP	20	17-AUG-97
Davies	ST_CLERK	50	29-JAN-97
Abel	SA_REP	80	11-MAY-96
King	AD_PRES	90	17-JUN-87

20 rows selected.

Сортировка по псевдониму столбца

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal;
```

EMPLOYEE_ID	LAST_NAME	ANNSAL
144	Vargas	30000
143	Matos	31200
142	Davies	37200
141	Rajs	42000
107	Lorentz	50400
200	Whalen	52800
124	Mourgos	69600
104	Ernst	72000
202	Fay	72000
178	Grant	84000
206	Gietz	99600
100	King	288000

20 rows selected.

Сортировка по нескольким столбцам

Последовательность сортировки определяется порядком столбцов в предложении **ORDER BY**.

Можно сортировать по столбцу, не входящему в **SELECT** список.

```
SELECT last_name, department_id, salary
FROM   employees
ORDER BY department_id, salary DESC;
```

LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Hartstein	20	13000
Fay	20	6000
Mourgos	50	5800
Rajs	50	3500
Higgins	110	12000
Gietz	110	8300
Grant		7000

20 rows selected.

Выводы по теме №2

- Использование предложения **WHERE** для ограничения количества выводимых строк:
  - Условия сравнения;
  - Условия **BETWEEN**, **IN**, **LIKE** и **NULL;**
  - Логические операторы **AND**, **OR** и **NOT.**
- Использование предложения **ORDER BY** для сортировки выходных результатов.

```
SELECT      *|{[DISTINCT] столбец|выражение [псевдоним],...}
FROM        таблица
[WHERE      условие(я)]
[ORDER BY   {столбец, выражение, псевдоним} [ASC|DESC]];
```

Контрольные задания по теме №2

1. Создайте запрос отображающий фамилию и продажи сотрудников, зарабатывающих более $12000. Сохраните SQL в текстовом файле lab2_1.sql. Выполните запрос.
2. Создайте запрос отображающий фамилию и департамент работника номер 176.
3. Измените lab2_1.sql так, что бы вывести фамилию и продажи всех

сотрудников чьи продажи не попадают в диапазон от $5000 до $12000. Сохраните SQL в текстовом файле lab2_3.sql.

4. Отобразите фамилию, job_id, и дату начала работы (hire_date) для сотрудников устроившихся на работу между 20 февраля 1998 года и 1 мая 1998 года. Отсортируйте результат по hire_date в порядка возрастания.

5. Отобразите фамилию и номер департамента для всех сотрудников из 20 и 50 департамента, сортируя в алфавитном порядке по имени.

6. Измените lab2_3.sql что бы отобразить фамилию и продажи сотрудников, которые зарабатывают между $5000 и $12000, и работают в 20 или 50 департаменте. Назовите столбцы ***Employee*** и ***Monthly Salary***, соответственно. Сохраните запрос в файле lab2_6.sql, и запустите.

7. Отобразите фамилию и дату приема на работу, для каждого сотрудника, принятого на работу в 1994 году.

8. Отобразите фамилию и job_id для всех сотрудников не имеющих начальника.

9. Отобразите фамилию, продажи и комиссионные для всех сотрудников получающих комиссионные. Отсортируйте данные в порядке убывания продаж и комиссионных.

10. Отобразите фамилию всех сотрудников у которых третья буква фамилии "***a***".

11. Отобразите фамилию всех сотрудников у которых в фамилии есть буквы "***a***" и "***e***".

12. Отобразите фамилию, job_id и продажи для всех сотрудников чья работа (job_id)SA_REP или ST_CLERK, и чьи продажи не равны $2500, $3500 или $7000.

13. Измените lab2_6.sql так, что бы отобразить фамилию, продажи и комиссионные для всех сотрудников, чьи комиссионные равны 20%. Сохраните запрос как lab2_13.sql, и запустите его.

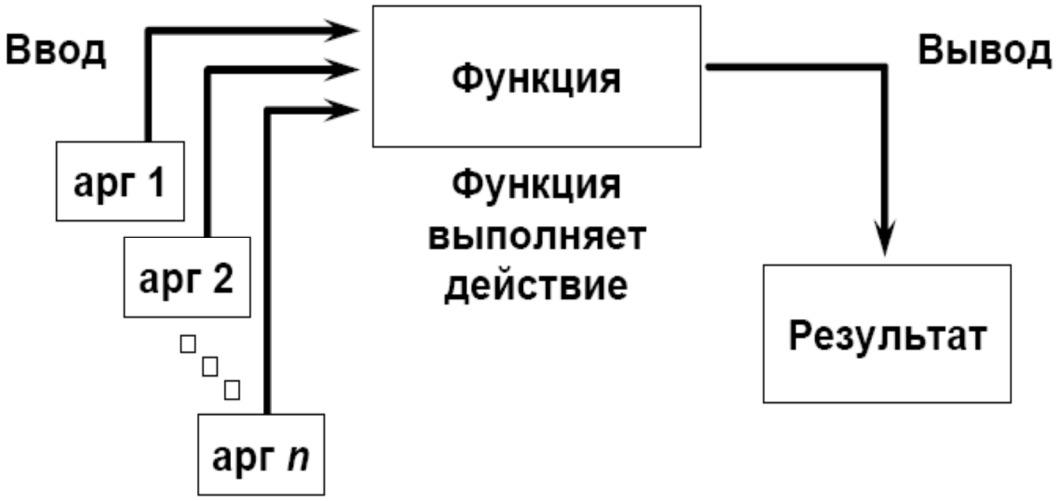
ТЕМА 3. ОДНОСТРОЧНЫЕ ФУНКЦИИ

Рассматриваемые вопросы:

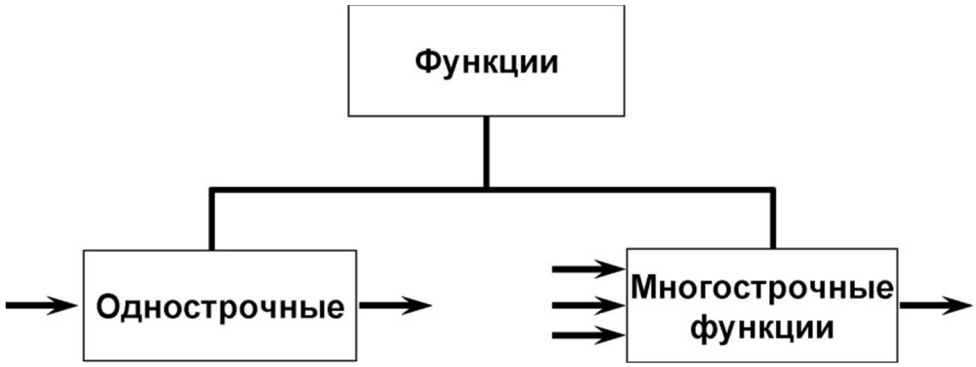
- Различные типы функций в SQL.
- Использование функций различных типов: символьных, числовых и типа «дата» в командах **SELECT**.
- Функции преобразования данных и их использование.
- Составление запросов, требующих использования числовых, символьных функций и функций для работы с датами.
- Использование конкатенации с функциями.
- Составление запросов, нечувствительных к регистру символов, для проверки полезности символьных функций.

Более детально с вопросами темы 3 в плане теории можно самостоятельно ознакомиться в источниках [3], [5] и [6].

Функции SQL



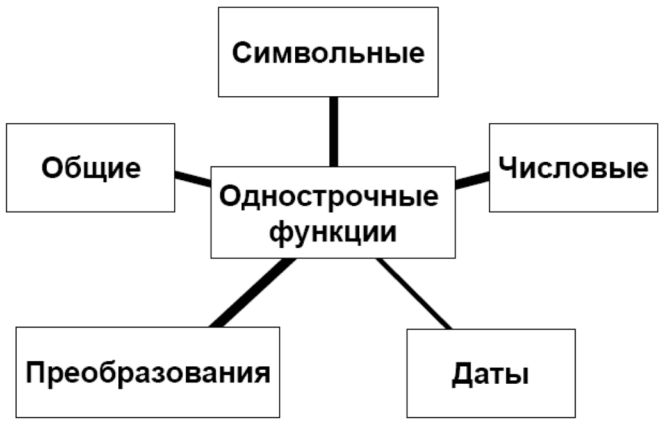
Функций SQL делятся на два типа:

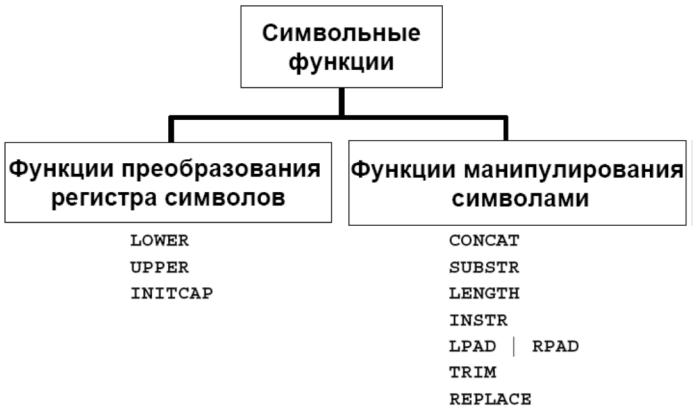


Однострочные функции:

- Манипулируют элементами данных.
- Принимают аргументы и возвращают одно значение.
- Работают с каждой строкой, возвращаемой запросом.
- Возвращают один результат на строку.
- Могут изменять тип данных.
- Могут быть вложенными.
- Принимают аргументы, которые могут быть столбцами или выражениями.

```
имя_функции [(арг1, арг2,...)]
```





Функция	Описание
LOWER(столбец|выражение)	Меняет регистр символов на нижний
UPPER(столбец|выражение)	Меняет регистр символов на верхний
INITCAP(столбец|выражение)	Изменяет регистр первой буквы каждого слова на верхний, а остальные на нижний
CONCAT(столбец1|выражение1, столбец2|выражение2)	Объединяет первое символьное значение со вторым; является эквивалентом оператора объединения (||)
SUBSTR(столбец|выражение, m[, n])	Возвращает набор символов из заданной строки, начиная с позиции m. (Если m отрицательное число, то отсчет начинается с конца стоки. Если n опущено, то возвращаются все символы до конца строки)
LENGTH(столбец|выражение)	Возвращает количество символов в выражении
INSTR(столбец|выражение[, m][, n])	Возвращает номер позиции строки. Дополнительно Вы можете указать m – позиция, с которой начинается поиск, и n – номер нахождения. По умолчанию m и n равны 1, это означает, что поиск стартует в начале и выводится первое нахождение.

Функция	Описание
LPAD(столбец|выражение, n, 'строка')	Добавляется с лева к столбцу|значению выражения строка так, что бы общая длина была n.
RPAD(столбец|выражение, n, 'строка')	Добавляется с права к столбцу|значению выражения строка так, что бы общая длина была n.
TRIM(символ FROM источник)	Вырезает символ из источника
REPLACE(текст, искомая строка, строка замены)	Заменяет в тексте искомую строку на строку замены

Функции преобразования регистра символов

Эти функции преобразуют регистр символьных строк

Функция	Результат
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

Пример. Использования функций преобразования регистра: вывод номера служащего по фамилии Higgins, его фамилии и отдела:

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  last_name = 'higgins';
no rows selected
```

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
205	Higgins	110

Функции манипулирования символами

Эти функции манипулируют символьными строками:

Функция	Результат
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
TRIM('H' FROM 'HelloWorld')	elloWorld

Использование функций манипулирования символами

```
SELECT employee_id, CONCAT(first_name, last_name) NAME, job_id,
       LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(job_id, 4) = 'REP';
```

EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
174	EllenAbel	SA_REP	4	0
176	JonathonTaylor	SA_REP	6	2
178	KimberelyGrant	SA_REP	5	3
202	PatFay	MK_REP	3	2

Числовые функции

- ROUND: округляет значение до заданной точности

ROUND(45.926,2) → 45.93

- TRUNC: усекает значение до заданного количества десятичных знаков

TRUNC(45.926,2) → 45.92

- MOD: возвращает остаток от деления

MOD(1600,300) → 100

Использование функции ROUND

```
SELECT ROUND(45.923,2), ROUND(45.923,0),
       ROUND(45.923,-1)
FROM   DUAL;
```

ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
45.92	46	50

DUAL – это фиктивная таблица, которую можно использовать для получения результатов выполнения функций и вычислений.

Использование функции TRUNC

```
SELECT  TRUNC(45.923,2), TRUNC(45.923),
        TRUNC(45.923,-2)
FROM    DUAL;
```

TRUNC(45.923,2)	TRUNC(45.923)	TRUNC(45.923,-2)
45.92	45	0

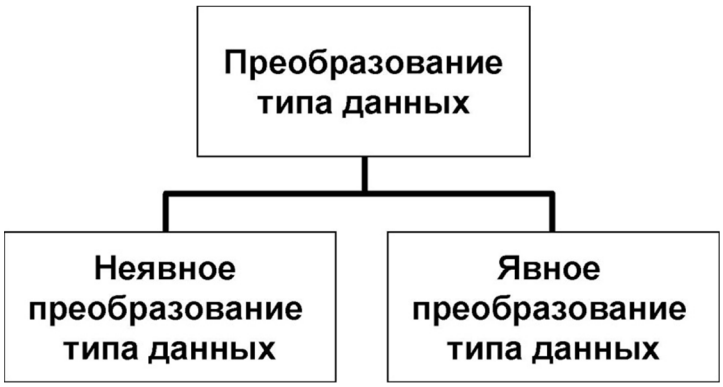
Использование функции MOD

Вычисление остатка отделения оклада на 5000 для всех служащих, работающих в должности торгового представителя.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM   employees
WHERE  job_id = 'SA_REP';
```

LAST_NAME	SALARY	MOD(SALARY,5000)
Abel	11000	1000
Taylor	8600	3600
Grant	7000	2000

Функции преобразования



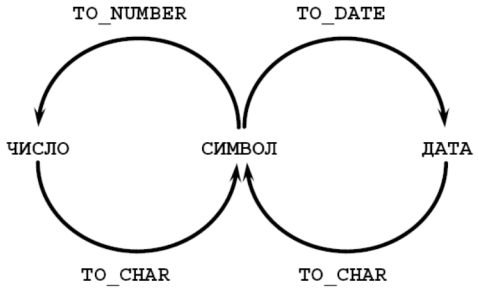
Неявное преобразование типов данных

Для операций присваивания Oracle может автоматически выполнять следующие преобразования:

При вычислении выражений Oracle может автоматически выполнять следующие преобразования:

Исходный формат	Новый формат
VARCHAR2 или CHAR	NUMBER
VARCHAR2 или CHAR	DATE

Явное преобразование типов данных



Функция	Описание
TO_CHAR(число|дата[, *ftm*][, *nlsparams*])	Преобразует число или дату в строку типа VARCHAR2, используя формат *ftm*. **Преобразование чисел**: Параметр *nlsparam* определяет текущие характеристики, которые возвращаются элементами числового формата: • Разделитель дробной части • Разделитель групп • Символ местной валюты • Международные символы валют. Если *nlsparams* или другие параметры опущены, то функция использует параметры по умолчанию для сессии. Функцией возвращаются специфические для языка названия месяцев, дней и их аббревиатуры. Если этот параметр опущен, то функция использует языковые настройки сессии.
TO_NUMBER(строка[, *ftm*][, *nlsparams*])	Преобразует строку символов, содержащую цифры, в число в формате, определенном параметром *ftm*. Параметр *nlsparams* имеет то же назначение, что и в функции TO_CHAR.
TO_DATE(строка[, *ftm*][, *nlsparams*])	Преобразует символьную строку представляющую дату в тип данных дата соответственно параметру *ftm*. Если *ftm* опущен, то формат DD-MON-YY. Параметр *nlsparams* имеет то же назначение, что и в функции TO_CHAR.

Функция TO_CHAR с датами

```
TO_CHAR(дата, 'модель_формата')
```

Модель формата:

- Должна быть заключена в апострофы. Различает символы верхнего и нижнего регистров.
- Может включать любые разрешенные элементы формата даты.
- Использует элемент fm для удаления конечных пробелов и ведущих нулей.
- Отделяется от значения даты запятой

Элементы формата даты

YYYY	Полный год цифрами
YEAR	Год прописью
MM	Двузначное цифровое обозначение месяца
MONTH	Полное название месяца
DY	Трехзначное алфавитное сокращенное название дня недели
DAY	Полное название дня недели
DD	Номер дня месяца

Элементы модели формата даты

• Элементы, которые задают формат части даты, обозначающей время.

HH24:MI:SS → 15:45:32

• Символьные строки добавляются в кавычках.

DD "of" MONTH → 12 of OCTOBER

• Числовые суффиксы используются для вывода числительных прописью.

ddspth → fourteenth

Элементы формата даты: формат времени

Элемент	**Описание**
AM или PM	Индикатор полудня
A.M. или P.M.	Индикатор полудня с периодами
HH или HH12 или HH24	Часы или часы (1-12) или часы (0-23)
MI	Минуты (0-59)
SS	Секунды (0-59)
SSSSS	Количество секунд прошедших с полночи (0-86399)

Другие форматы

Элемент	**Описание**
/ . ,	Знаки препинания, отображаемые в результате
“текст”	Текс в двойных кавычках, отображаемая в результате

Специфические суффиксы отображения чисел

Элемент	**Описание**
TH	Порядковое число (на пример DDTH → 4TH)
SP	Произношение числа (на пример DDSP → FOUR)
SPTH или THSP	Произношение порядковых чисел (на пример DDSPTH → FOURTH)

Пример использования:

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY') HIREDATE
FROM   employees;
```

LAST_NAME	HIREDATE
King	17 June 1987
Kochhar	21 September 1989
De Haan	13 January 1993
Hunold	3 January 1990
Ernst	21 May 1991
Lorentz	7 February 1999
Mourgos	16 November 1999
Rais	17 October 1995
Gietz	7 June 1994

20 rows selected.

Использование функции TO_CHAR с числами

```
TO_CHAR(число, 'модель_формата')
```

Форматы, используемые с функцией TO_CHAR для вывода числового значения в виде символьной строки:

9	Цифра
0	Вывод нуля
$	Плавающий знак доллара
L	Плавающий символ местной валюты
.	Вывод десятичной точки
,	Вывод разделителя троек цифр

Пример использования:

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

SALARY
$6,000.00

Использование функций TO_NUMBER и TO_DATE

- Преобразование символьной строки в числовой формат с использованием функции TO_NUMBER:

```
TO_NUMBER(char[, 'модель_формата'])
```

- Преобразование символьной строки в формат даты с использованием функции TO_DATE:

```
TO_DATE(char[, 'модель_формата'])
```

- В этих функциях можно использовать модификатор fx. В функции TO_DATE он задает точное соответствие символьного аргумента и модели формата даты.

Формат даты RR

Текущий год	Заданная дата	Формат RR	Формат YY
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		Год, заданный двузначным числом	
		0–49	50–99
Если две последних цифры текущего года равны:	0–49	Возвращаемая дата относится к текущему столетию.	Возвращаемая дата относится к столетию перед текущим.
	50–99	Возвращаемая дата относится к столетию после текущего.	Возвращаемая дата относится к текущему столетию.

Пример формата даты RR

Чтобы найти сотрудников, принятых на работу до 1990 года, используйте формат RR. Выполнение команды даст одинаковый результат, независимо от того, когда выполнялась команда (сейчас или в 1999 году):

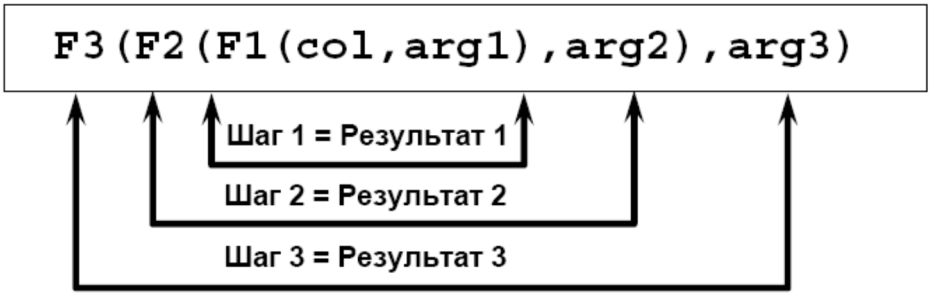
```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM   employees
WHERE  hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

LAST_NAME	TO_CHAR(HIR
King	17-Jun-1987
Kochhar	21-Sep-1989
Whalen	17-Sep-1987

Вложенные функции

- Однострочные функции могут быть вложены на любую глубину.

- Вложенные функции вычисляются от самого глубокого уровня к внешнему.




```
SELECT last_name,
       NVL(TO_CHAR(manager_id), 'No Manager')
FROM   employees
WHERE  manager_id IS NULL;
```

LAST_NAME	NVL(TO_CHAR(MANAGER_ID),'NOMANAGER')
King	No Manager

Данный запрос выводит главу компании, у которого нет руководителя. Определение результата SQL состоит из двух этапов:

1. Определение результата вычисления внутренней функции, преобразующей число в строку (Результат1 = TO_CHAR(manager_id))
2. Определение результата вычисления внешней функции, замещающей значение NULL на текстовую строку (NVL(Результат1, 'No Manager')).

Общие функции

Эти функции работают с любыми типами данных и обрабатывают неопределенные значения.

- NVL (выражение1, выражение2)
- NVL2 (выражение1, выражение2, выражение3)
- NULLIF (выражение1, выражение2)
- COALESCE (выражение1, выражение2, ..., выражениеn)

Функция	Описание
NVL	Заменяет NULL на фактическое значение
NVL2	Если выражение1не NULL, NVL2 возвращает выражение2. Если выражение1имеет NULL значение, то возвращается выражение3. Аргумент

	функции выражение1 может быть любого типа данных.
NULLIF	Сравнивает два выражения, и возвращает NULL если они равны, или выражение1 если они не равны
COALESCE	Возвращает первое не-NULL выражение из списка выражений.

Функция NVL

- Преобразует неопределенное значение в действительное
- Используемые типы данных – DATE, символьные (CHARACTER) и числовые (NUMBER).
- Типы данных должны совпадать:
  - NVL(commission_pct,0)
  - NVL(hire_date,'01-JAN-97')
  - NVL(job_id,'No Job Yet')

Использование функции NVL

```
SELECT last_name, salary, NVL(commission_pct, 0),
   (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
King	24000	0	288000
Kochhar	17000	0	204000
De Haan	17000	0	204000
Hunold	9000	0	108000
Ernst	6000	0	72000
Lorentz	4200	0	50400
Mourgos	5800	0	69600
Rajs	3500	0	42000
Davies	3100	0	37200
Matos	2600	0	31200
Vargas	2500	0	30000
Zlotkey	10500	.2	151200
Abel	11000	.3	171600

20 rows selected.

Использование функции NVL2

```
SELECT last_name,  salary, commission_pct,
       NVL2(commission_pct,
                    'SAL+COMM', 'SAL') income
FROM   employees WHERE department_id IN (50, 80);
```

LAST_NAME	SALARY	COMMISSION_PCT	INCOME
Zlotkey	10500	.2	SAL+COMM
Abel	11000	.3	SAL+COMM
Taylor	8600	.2	SAL+COMM
Mourgos	5800		SAL
Rajs	3500		SAL
Davies	3100		SAL
Matos	2600		SAL
Vargas	2500		SAL

8 rows selected.

Использование функции NULLIF

```
SELECT first_name, LENGTH(first_name) "expr1",
       last_name,  LENGTH(last_name)  "expr2",
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM   employees;
```

FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
William	7	Gietz	5	7
Shelley	7	Higgins	7	
Pat	3	Fay	3	
Michael	7	Hartstein	9	7
Jennifer	8	Whalen	6	8
Kimberely	9	Grant	5	9
Jonathon	8	Taylor	6	8
Ellen	5	Abel	4	5
Eleni	5	Zlotkey	7	5
Peter	5	Vargas	6	5
Randall	7	Matos	5	7
Curtis	6	Davies	6	
Trenna	6	Rajs	4	6

20 rows selected.

Использование функции COALESCE

Преимущество функции COALESCE по сравнению с функцией NVL состоит в том, что функция COALESCE может обрабатывать несколько альтернативных значений.

Если первое выражение определенно, функция возвращает это выражение; в противном случае она проверяет оставшиеся выражения.

```
SELECT   last_name,
         COALESCE(commission_pct, salary, 10) comm
FROM     employees
ORDER BY commission_pct;
```

LAST_NAME	COMM
Grant	.15
Zlotkey	.2
Taylor	.2
Abel	.3
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Matos	2600
Vargas	2500

20 rows selected.

Условные выражения

- Позволяют применять логические конструкции ЕСЛИ-ТО-ИНАЧЕ(IF-THEN-ELSE) внутри команды SQL
- Два метода:
  - выражение CASE
  - функция DECODE

Выражение CASE

Помогает создавать условные запросы, которые выполняют действия логического оператора IF-THEN-ELSE:

```
CASE выражение
          WHEN сравн_выражение1 THEN возвр_выражение1
         [WHEN сравн_выражение2 THEN возвр_выражение2
          WHEN сравн_выражениеn THEN возвр_выражениеn
          ELSE else_выражение]
END
```

Использование выражения CASE

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
                   WHEN 'ST_CLERK' THEN  1.15*salary
                   WHEN 'SA_REP'   THEN  1.20*salary
       ELSE       salary END
FROM   employees;
```

Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...	...	...	...
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

Функция DECODE

Помогает создавать условные запросы, которые выполняют действия логического условия CASE или оператора IF-THEN-ELSE .

```
DECODE(столбец|выражение, вариант1, результат1
                          [, вариант2, результат2,...,]
                          [, результат_по_умолчанию])
```

Использование функции DECODE

```
SELECT last_name, job_id, salary,
       DECODE(job_id, 'IT_PROG',  1.10*salary,
                      'ST_CLERK', 1.15*salary,
                      'SA_REP',   1.20*salary,
              salary)
       REVISED_SALARY
FROM   employees;
```

Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...	...	...	...
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

Пример. Показать ставку налога на заработную плату для сотрудников 80 отдела:

```
SELECT last_name, salary,
       DECODE (TRUNC(salary/2000, 0),
                          0, 0.00,
                          1, 0.09,
                          2, 0.20,
                          3, 0.30,
                          4, 0.40,
                          5, 0.42,
                          6, 0.44,
                             0.45) TAX_RATE
FROM   employees
WHERE  department_id = 80;
```

Данный пример показывает, как вычислить ставку налога для каждого сотрудника 80-ого департамента, основываясь на месячных продажах.

Месячные продажи	Ставка налога, %
$0,00 – 1999,99	00
$2000,00 – 3999,00	09
$4000,00 – 5999,99	20
$6000,00 – 7999,99	30
$8000,00 – 9999,99	40
$10000,00 – 11999,99	42
$12000,00 – 13999,99	44
$14000,00 и выше	45

Выводы по теме №3

С помощью функций осуществляются:

- Вычисления с данными;
- Изменение отдельных элементов данных;
- Манипулирование выводом групп строк;
- Изменение форматов дат для вывода;
- Преобразование формата данных столбцов;
- Обработка неопределенных значений;
- Логическая обработка IF-THEN-ELSE, которая может быть также выполнена с помощью условного выражение CASE.

Контрольные задания по теме №3

1. Напишите запрос, отображающий текущую дату. Назовите столбец Date.

2. Для каждого сотрудника выведите employee_id, last_name, salary и salary увеличенное на 15%, округлив до целого числа. Назовите столбец New Salary. Сохраните SQL запрос в файл lab3_2.sql.

3. Запустите запрос из файла lab3_2.sql.

4. Отредактируйте запрос из файла lab3_2.sql так чтобы добавить столбец, в котором вычитается salary из new salary. Назовите новый столбец Increase. Сохраните SQL запрос в файл lab3_4.sql. Выполните измененный зарос.

5. Напишите запрос, который отображает фамилии сотрудников (last_name) так, что первая буква фамилии заглавная, а остальные прописные и длину фамилии, для всех сотрудников, чьи фамилии начинаются на *J*, *A* или *M*. Дайте каждому столбцу соответствующее имя. Рассортируйте результат по фамилии.

6. Для каждого сотрудника отобразите фамилию и посчитайте количество месяцев между сегодняшним днем и датой приема на работу (hire_date). Назовите столбец MONTHS_WORKED. Отсортируйте результат по количеству месяцев, которые проработал сотрудник. Округлите количество месяцев к ближайшему целому числу.

7. Напишите запрос который будет выводить для каждого сотрудника такую строку: <last_name> earns <salary> monthly but wants <3*salary>. Назовите столбец Dream Salaries.

8. Напишите запрос, отображающий фамилии и продажи (salary) для всех сотрудников. Длина строки продаж должна быть 15. Если длина меньше, добавляйте слева знак $ до нужной длины.

9. Отобразите фамилию, дату приема на работу и дату проверки продаж – это первый понедельник после 6-ти месяцев работы. Назовите столбец REVIEW. Формат даты должен быть подобен следующему “Monday, the Thirty-First of July, 2000”.

10. Отобразите фамилию, дату приема на работу и день недели, с которого начал работать сотрудник. Назовите столбец Day. Отсортируйте

результат по дню недели, с которого начал работать сотрудник, начиная с понедельника.

11. Создайте запрос, отображающий фамилию и комиссионные (commission_pct). Если сотрудник не получает комиссионных, напишите “No Commission”. Назовите столбец COMM.

12. Создайте запрос, отображающий фамилию и индикатор его годовых продаж (*). Каждая звездочка (*) обозначает одну тысячу долларов. Отсортируйте данные в обратном порядке продаж. Назовите столбец EMPLOYEES_AND_THEIR_SALARIES.

13. Используя функцию DECODE, напишите запрос, отображающий ранг всех сотрудников, основываясь на значении столбца JOB_ID, используя следующую таблицу:

JOB_ID	**Ранг**
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
Другое	0

14. Переделайте предыдущее задание, используя синтаксис оператора CASE.

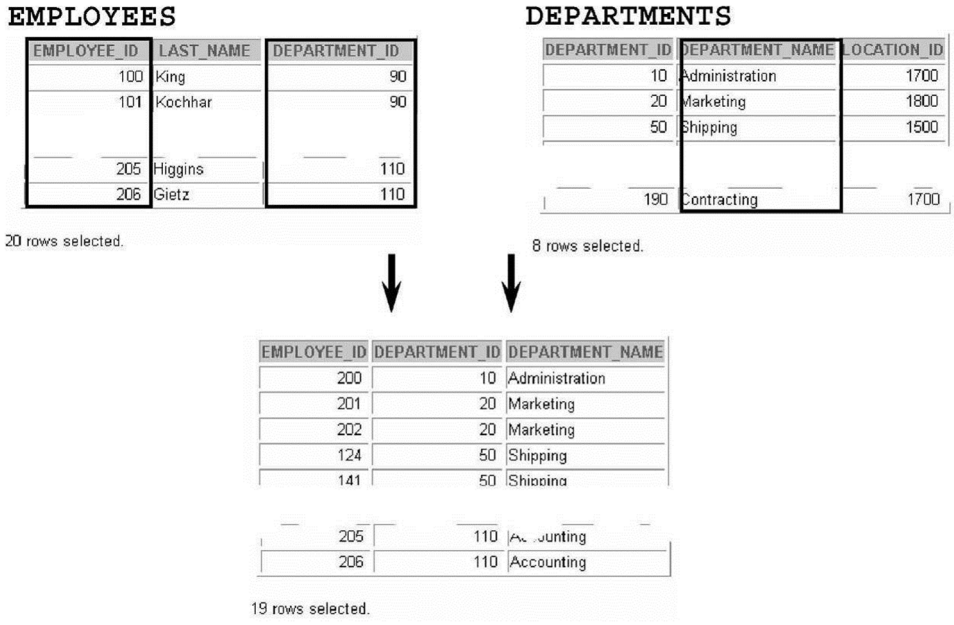
ТЕМА 4. ВЫБОРКА ДАННЫХ ИЗ НЕСКОЛЬКИХ ТАБЛИЦ

Рассматриваемые вопросы:

- Команды **SELECT** для выборки данных из нескольких таблиц, с помощью эквисоединений и прочих видов соединений.
- Использование внешних соединений для просмотра данных, неудовлетворяющих обычным условиям соединения.
- Соединение таблицы с собой.
- Соединение таблиц с использованием эквисоединения
- Выполнение внешних соединений и соединений таблицы с собой.
- Включение дополнительных условий.

Более детально с вопросами темы 4 в плане теории можно самостоятельно ознакомиться в источнике [1, глава 6].

Выборка данных из нескольких таблиц



Иногда нужно использовать данные более чем из одной таблицы. В примере показан отчет, отображающий данные из двух разных таблиц.

- Employee ID есть в таблице EMPLOYEES

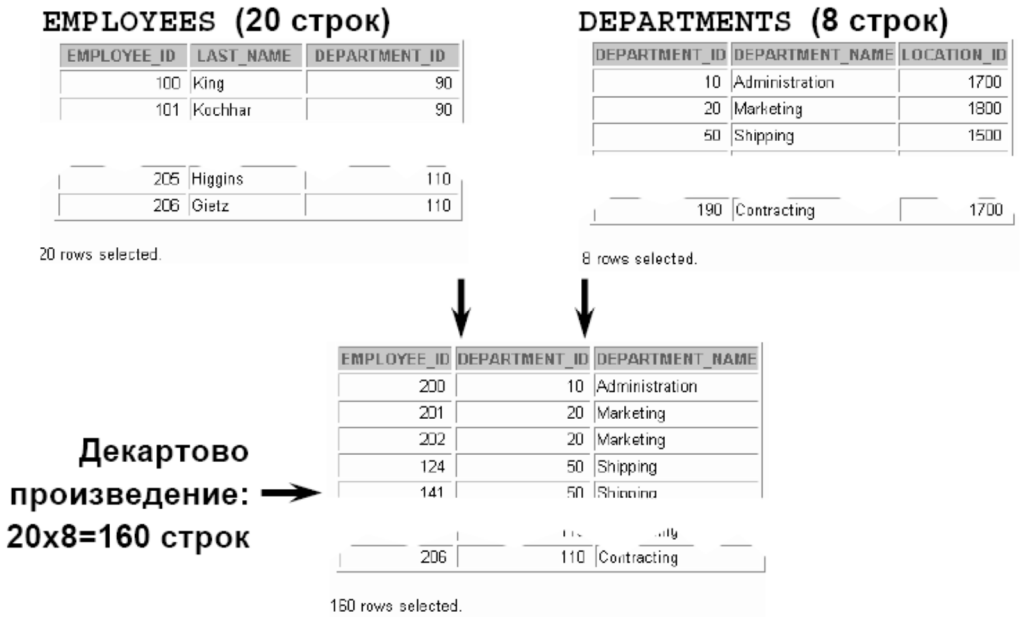
- Department ID есть в обоих таблицах: EMPLOYEES и DEPARTMENTS
- Location ID есть в таблице DEPARTMENTS.

Для получения такого отчета необходимо связать таблицы EMPLOYEES и DEPARTMENTS, и получить данные из них.

Декартово произведение

- Декартово произведение образуется, если:
  - опущено условие соединения;
  - условие соединения не действительно;
  - все строки первой таблицы соединяются со всеми строками второй таблицы.
- Во избежание получения декартова произведения предложение **WHERE** всегда должно включать правильное условие соединения.

Получение декартова произведения



Когда условие соединения не верно или опущено результатом исполнения запроса является декартово произведение, в котором отображаются комбинации всех строк. Т.е. все строки первой таблицы соединяются со всеми строками из второй таблицы.

Декартово произведение имеете тенденцию к созданию большого количества строк, и его результат очень редко может быть применим. Вы

всегда должны включать правильное условие соединения в условие **WHERE**, до тех пор кока не определите необходимое для объединения всех строк из всех таблиц.

Декартово произведение применяется для некоторых тестов, когда необходимо сгенерировать большое количество строк для имитации необходимого количества данных.

Виды соединений

Разработанные Oracle соединения (версии до 8i включительно)	SQL: 1999 Различные типы соединений
1. Эквисоединение 2. Не-эквисоединение 3. Внешнее соединение 4. Соединение таблицы с собой	1. Перекрестные соединения 2. Натуральные соединения 3. опция USING 4. Полное или двухстороннее внешнее соединение 5. Произвольные условия соединения для внешнего соединения

Соединение таблиц с использованием синтаксиса, разработанного Oracle

Соединение используется для выборки данных из нескольких таблиц.

```
SELECT    таблица1.столбец, таблица2.столбец
FROM      таблица1, таблица2
WHERE     таблица1.столбец1 = таблица2.столбец2;
```

Условие соединения указывается в предложении **WHERE**. Если одно и тоже имя столбца присутствует более чем водной таблице, к имени столбца добавляется имя таблицы в виде префикса.

Когда требуются данные более чем из одной таблицы базы данных используется условие соединения. Строки из одной таблицы могут быть объединены со строками из другой таблицы согласно общим значениям в соответствующих столбцах. Обычно это первичный/главный (primary key) и внешний (foreign key) ключи.

Эквисоединение

Для определения названия департамента сотрудника сравниваются значения в столбце DEPARTMENT_ID таблицы EMPLOYEES с значением DEPARTMENT_ID таблицы DEPARTMENTS. Взаимоотношение между таблицами EMPLOYEES и DEPARTMENTS – эквисоединение, эти значения столбца DEPARTMENT_ID в обеих таблицах должны быть равны.

Заметка: Эквисоединение также называют простое соединение или внутренне соединение.

Выборка записей с помощью эквисоединений

```
SELECT employees.employee_id, employees.last_name,
       employees.department_id, departments.department_id,
       departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
205	Higgins	110	110	1700
206	Gietz	110	110	1700

19 rows selected.

В этом примере:

Предложение SELECT определяет выбираемые столбцы:

- Фамилию сотрудника, порядковый номер сотрудника и номер департамента, которые находятся в таблице EMPLOYEES
- Номер департамента, название департамента и LOCATION_ID которые находятся в таблице DEPARTMENTS.

Предложение FROM определяет две таблицы, которым должна обратиться база данных для нахождения требуемой информации:

- Таблица EMPLOYEES
- Таблица DEPARTMENTS.

Предложение WHERE определяет как будут объединены таблицы: EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID.

Поскольку столбец DEPARTMENT_ID есть в обеих таблицах, то перед ним нужно поставить префикс таблицы, что бы избежать неопределенности.

Дополнительные условия поиска с оператором AND

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
Whalen	10
Hartstein	20
Fay	20
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Hunold	60
Ernst	60
Lorentz	60
Zlotkey	80
Dietz	110

19 rows selected.

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT
60	IT
80	Sales
110	Accounting

В дополнение к объединению возможны критерии в предложении WHERE, что бы отобрать строки из объединения для одной или более таблиц в объединении. Например, что бы отобразить название департамента и его номер, в котором работает Matos, необходимо дополнительное условие в предложении WHERE.

```
SELECT last_name, employees.department_id,
                department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND last_name = 'Matos';
```

Различение столбцов с одинаковыми именами

- Для различения одноименных столбцов из разных таблиц используются префиксы в виде имен таблиц.
- Использование префиксов в виде имен таблиц увеличивает производительность.
- Одноименные столбцы из разных таблиц можно различать по их псевдонимам.
- Упрощает запросы.
- Повышает производительность.

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

Соединение более чем двух таблиц

Иногда возникает необходимость объединить более двух таблиц. Например, для отображения фамилии, названия департамента и города для каждого сотрудника необходимо объединить таблицы EMPLOYEES, DEPARTMENTS и LOCATIONS.

SELECT e.last_nast, d.department_name, l.city

FROM employees e, departments d, locations l

WHERE e.department_id=d.department_id

AND d.location_id = l.location_id;

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Grant	
Whalen	10
Hartstein	20
Fay	20
Higgins	110
Gietz	110

20 rows selected.

DEPARTMENTS

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

LOCATIONS

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

Для соединения n таблиц требуется, по крайней мере, (n-1) условий соединения. Следовательно, для соединения трех таблиц требуются, по крайней мере, два условия.

Не-эквисоединения

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
ray	00
Higgins	12000
Gietz	8300

20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

6 rows selected.

Оклад в таблице EMPLOYEES находится между нижней и верхней границами окладов в таблице JOB_GRADES

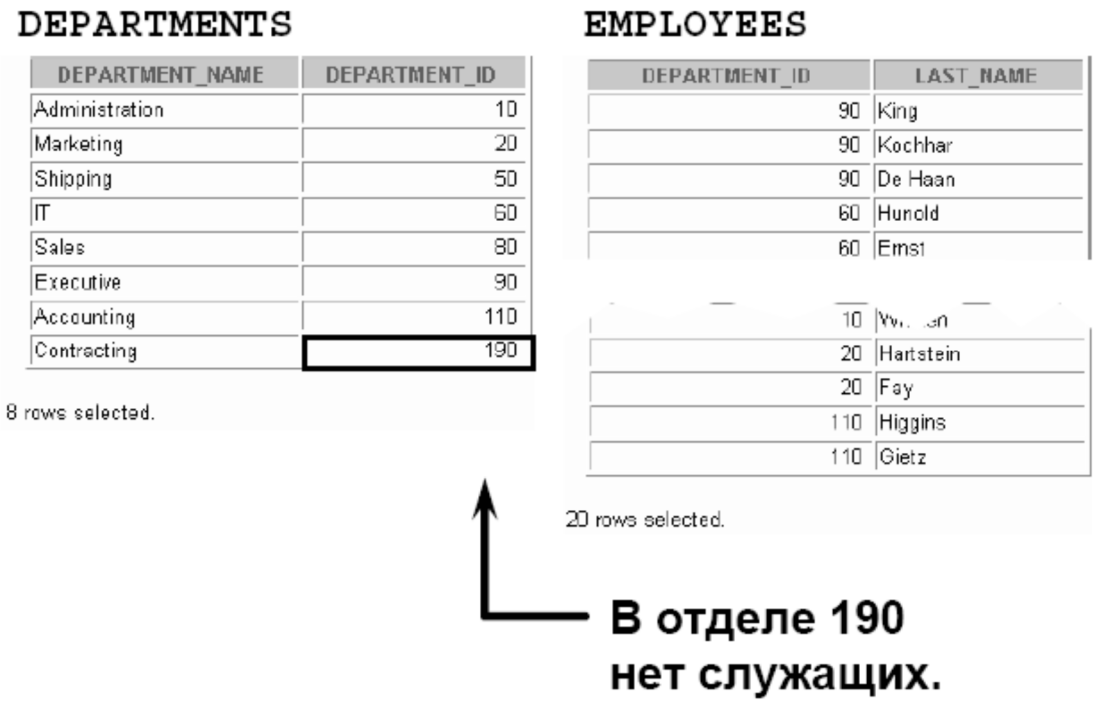
Не-эквисоединение это условие объединения содержащее нечто отличное от оператора равенства. Связь между таблицами EMPLOYEES и JOB_GRADES – это пример не-эквисоединения. Связь между таблицами такова, что значения столбца SALARY таблицы EMPLOYEES должно быть между значениями столбцов LOWEST_SAL и HIGHEST_SAL таблицы JOB_GRADES. Связь образуется с помощью оператора сравнения, отличного от оператора равенства (=).

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
kochhar	17000	E
De Haan	17000	E

20 rows selected.

Внешние соединения



Если трока не удовлетворяет условию объединения, то этой строки не будет в результате вывода. Например, в результате эквисоединения таблиц EMPLOYESS и DEPARTMENTS не будет сотрудника Grant, т.е. у него нет записи в столбце department_id таблицы EMPLOYESS. Поэтому вместо 20 записей будет выведено только 19.

Синтаксис внешнего соединения

- Внешнее соединение используется для выборки строк, не удовлетворяющих обычным условиям соединения.
- Оператором внешнего соединения является знак плюс (+).

```
SELECT таблица1.столбец, таблица2.столбец
FROM   таблица1, таблица2
WHERE  таблица1.столбец(+) = таблица2.столбец;
```

```
SELECT таблица1.столбец, таблица2.столбец
FROM   таблица1, таблица2
WHERE  таблица1.столбец = таблица2.столбец(+);
```

Использование внешних соединений

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id(+) = d.department_id;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
ggins	110	
Gietz	110	Accounting
		Contracting

20 rows selected.

Соединение таблицы с собой

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
101	Kochhar	100
102	De Haan	100
124	Mourgos	100
149	Zlotkey	100
201	Hartstein	100
200	Whalen	101
206	Gietz	205

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
100	King
100	King
100	King
100	King
101	Kochhar
205	Higgin...

19 rows selected.

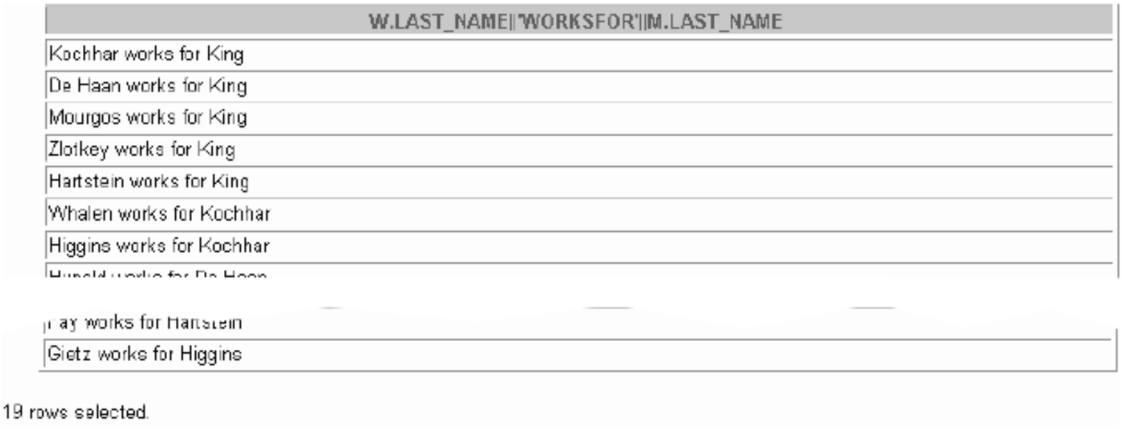
MANAGER_ID в таблице WORKER равен EMPLOYEE_ID в таблице MANAGER .

Для нахождения имени начальника каждого сотрудника необходимо соединить таблицу EMPLOYEES саму с собой. Например, что бы найти имя начальника сотрудника Whalen необходимо:

1. Найти Whalen в таблице EMPLOYEES по столбцу LAST_NAME.
2. Найти номер начальника для Whalen используя столбец MANAGER_ID. Номер начальника Whalen – 101.
3. Найти имя начальника по столбцу EMPLOYEE_ID=101,

производя поиск по столбцу LAST_NAME. Kochhar является сотрудником номер 101, т.е. Kochhar начальник Whalen.

```
SELECT w.last_name || ' works for '
       || m.last_name
FROM   employees w, employees m
WHERE  w.manager_id = m.employee_id;
```



W.LAST_NAME||'WORKSFOR'||M.LAST_NAME
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
ay works for Hartstein
Gietz works for Higgins

19 rows selected.

Соединение таблиц с использованием синтаксиса стандарта SQL: 1999

Используйте соединение для запроса информации из нескольких таблиц

```
SELECT   таблица1.столбец, таблица2.столбец
FROM     таблица1
[CROSS JOIN таблица2] |
[NATURAL JOIN таблица2] |
[JOIN таблица2 USING (имя_столбца)] |
[JOIN таблица2
  ON(таблица1.имя_столбца = таблица2.имя_столбца)] |
[LEFT|RIGHT|FULL OUTER JOIN таблица2
  ON (таблица1.имя_столбца = таблица2.имя_столбца)];
```

Создание перекрестных соединений

- Предложение **CROSS JOIN** используется для получения векторного произведения двух таблиц.
- Это то же самое, что и декартово произведение двух таблиц.

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments;
```

LAST_NAME	DEPARTMENT_NAME
Fay	Contracting
Higgins	Contracting
Gietz	Contracting

160 rows selected.

Создание натуральных соединений

- Предложение **NATURAL JOIN** основывается на всех столбцах двух таблиц, имеющих одинаковые имена.
- Выбираются строки из двух таблиц, которые имеют одинаковые значения во всех соответствующих столбцах.
- Если столбцы с одинаковыми именами имеют разные типы данных, возвращается сообщение об ошибке.

Выборка записей с помощью натуральных соединений

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations;
```

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

8 rows selected.

Создание соединений с использованием предложения USING

- Если несколько столбцов имеют одинаковые имена, но разные типы данных, предложение **NATURAL JOIN** может быть

заменено предложением **USING**, в котором можно явно указать столбцы, по которым нужно производить соединение.

- Имена таблиц и псевдонимы не указываются в ссылках на столбцы.
- Предложения **NATURAL JOIN** и **USING** – взаимно исключают друг друга.

Выборка записей с использованием предложения USING

```
SELECT e.employee_id, e.last_name, d.location_id
FROM  employees e JOIN departments d
USING (department_id);
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID
200	Whalen	1700
201	Hartstein	1800
202	Fay	1800
124	Mourgos	1500
141	Rajs	1500
142	Davies	1500
205	Higgins	1700
206	Gietz	1700

19 rows selected.

Создание соединений с помощью предложения ON

- В основе натурального соединения лежит эквисоединение всех столбцов с одинаковыми именами.
- Предложение **ON** используется для определения произвольных соединений или столбцов, участвующих в соединении.
- Отделяются условия соединения от условий ограничения.
- Предложение **ON** делает код более легким для понимания.

Выборка записей с использованием предложения ON

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
205	Higgins	110	110	1700
206	Gietz	110	110	1700

19 rows selected.

Создание трех сторонних соединений при помощи предложения ON

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
100	Seattle	Executive
101	Seattle	Executive
102	Seattle	Executive
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
206	Seattle	

19 rows selected.

Сравнение внутреннего (INNER) и внешнего (OUTER) соединений

- В соответствие со стандартом SQL: 1999 внутреннее соединение – это соединение двух таблиц, возвращающее только строки, соответствующие условию соединения.
- Соединение двух таблиц, возвращающее как строки внутреннего соединения, так и несоответствующие строк и левой (правой) таблицы, – это левое (правое) внешнее соединение.
- Полное внешнее соединение возвращает результаты внутреннего соединения, а также левого и правого внешнего соединений.

Oracle	SQL: 1999
Эквисоединение	Натуральное или внутренне соединение
Внешнее соединение	Левое внешнее соединение
Соединение с собой	Соединение ON
Не-эквисоединение	Соединение USING
Декартово произведение	Перекрестное соединение

Левое внешнее соединение

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
King	90	Executive
Kochhar	90	Executive
Ernst	60	IT
Grant		
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Higgins	110	Accounting
Gietz	110	Accounting

20 rows selected.

Правое внешнее соединение

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
Gietz	110	Accounting
		Contracting

20 rows selected.

Полное внешнее соединение

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
FULL OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Abel	80	Sales
Davies	50	Shipping
De Haan	90	Executive
Ernst	60	IT
Fay	20	Marketing
Gietz	110	Accounting
Grant		
Hartstein	20	Marketing
Zlotkey	80	
		Contracting

21 rows selected.

Полное внешнее соединение выводит все строки из всех таблиц для которых не было найдено соответствие.

Дополнительные условия

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
AND    e.manager_id = 149;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
174	Abel	80	80	2500
176	Taylor	80	80	2500

Выводы по теме №4

Синтаксис соединений, который был разработан Oracle для версий 8i и более ранних версий;

Применение стандартного синтаксиса SQL99 в версии 9i;

Соединение таблиц с использованием эквисоединения;

Выполнение внешних соединений и соединений таблицы с собой;

Включение дополнительных условий.

Контрольные задания по теме №4

1. Напишите запрос, отображающий фамилию, номер департамента и название департамента (department_name) для всех сотрудников.

2. Выведите список уникальных работ в департаменте 90. В вывод включите идентификатор местонахождения департамента (location_id).

3. Напишите запрос для отображения фамилии, названия департамента, идентификатор местонахождения и город сотрудников зарабатывающих комиссионные.

4. Отобразите фамилии и названия департаментов всех сотрудников фамилия которых содержит строчную букву “a”. Сохраните текст запроса в файл lab4_4.sql.

5. Напишите запрос для отображения фамилии, типа работы, номера департамента и названия департамента всех сотрудников работающих в Торонто (Toronto).

6. Отобразите фамилии и идентификаторы сотрудников вместе с фамилиями и идентификаторами их начальников. Назовите столбцы Employee, Emp#, Manager и Mgr#, соответственно. Сохраните текст запроса в файл lab4_6.sql.

7. Измените lab4_6.sql так, что бы вывод отображал всех сотрудников, включая King’a, который не имеет начальника. Сохраните текст запроса в файл lab4_7.sql.

8. Напишите запрос, отображающий фамилию, номер департамента и всех сотрудников, которые работают с ним в одном департаменте (в каждой строчке по одному коллеге). Дайте столбцам соответствующие названия.

9. Отобразите структуру таблицы JOB_GRADES. Создайте запрос для отображения фамилии, типа работы, названия департамента, продаж и ранг всех сотрудников.

10. Создайте запрос, отображающий фамилию и дату приема на работу сотрудников принятых на работу после сотрудника Davies.

11. Создайте запрос, отображающий фамилию и дату приема на работу сотрудников принятых на работу раньше их начальника. Так же отобразите фамилию и дату приема на работу начальника. Назовите столбцы Employee, Emp Hired, Manager и Mgr Hired, соответственно.

ТЕМА 5. АГРЕГИРОВАНИЕ ДАННЫХ С ПОМОЩЬЮ ГРУППОВЫХ ФУНКЦИЙ

Рассматриваемые вопросы:

- Общие сведения об имеющихся групповых функциях;
- Использование групповых функций;
- Вывод данных по группам с помощью предложения **GROUP BY**;
- Включение и исключение групп с помощью предложения **HAVING**;
- Демонстрация запросов с использованием групповых функций;
- Разбиение строк на группы для получения более чем одного результата.

Что такое групповые функции?

В противоположность однострочным функциям групповые функции работают с множеством строк (наборы) и возвращают один результат на группу. Этими наборами могут быть как таблица целиком, так и таблица, разбитая на группы.

Запросы могут производить обобщённое групповое значение полей точно так же, как и значение одного поля. Это делается с помощью агрегатных функций, или групповых функций. Агрегатные функции производят одиночное значение для всей группы таблицы.

Типы групповых функций

Функция	**Описание**
AVG([DISTINCT | ALL] n)	Среднее значение n, игнорируя null-значения
COUNT({* | [DISTINCT | ALL] *выражение*})	Количество строк, где *выражение* отлично от null (подсчет всех выбранных строк с помощью *, включая повторения и строки с null-значениями)
MAX([DISTINCT | ALL] *выражение*)	Максимальное значение *выражения*, игнорируя null-значения

Функция	**Описание**
MIN([DISTINCT | ALL] *выражение*)	Минимальное значение *выражения*, игнорируя null-значения
STDDEV([DISTINCT | ALL] n)	Стандартное отклонение n, игнорируя null-значения
SUM([DISTINCT | ALL] n)	Сумма значений n, игнорируя null-значения
VARIANCE([DISTINCT | ALL] n)	Отклонение от n, игнорируя null-значения

Агрегатные функции используются, подобно именам полей в предложении **SELECT**-запроса, но с одним исключением: они берут имена полей как аргументы. Только числовые поля могут использоваться с **SUM** и **AVG**. С функциями **COUNT, MAX и MIN** могут использоваться и числовые, и символьные поля.

При использовании с символьными полями, **MAX** и **MIN** будут транслировать их в эквивалент ASCII, который должен сообщать, что **MIN** будет означать первое, а **MAX** последнее значение в алфавитном порядке

DISTINCT делает функцию рассматривающей только неповторяющиеся значения; ALL делает функцию рассматривающей все значение, включая повторяющиеся.

Тип данных для функций с *выражением* может быть CHAR, VARCHAR2, NUMBER или DATE.

Все групповые функции игнорируют null-значения. Для замены null-значений используйте функции NVL, NVL2 или COALESCE.

Если используется предложение GROUP BY, то по умолчанию Oracle Server сортирует результат по возрастанию. Для изменения порядка сортировки может быть использовано DESC в предложении ORDER BY.

Синтаксис групповых функций

```
SELECT        [столбец,] групп_функция(столбец), ...
FROM          таблица
[WHERE        условие]
[GROUP BY     столбец]
[ORDER BY     столбец];
```

Использование функций AVG, MAX, MIN, SUM

Функции **AVG** и **SUM** применяются к столбцам с числовыми данными

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

Функции **MAX** и **MIN** применяются к данным любого типа

```
SELECT MIN(hire_date), MAX(hire_date)
FROM   employees;
```

MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

Данный запрос отображает самого молодого и самого старшего сотрудников.

Использование функции COUNT

Функция **COUNT** несколько отличается от всех остальных. Она считает число значений в данном столбце или число строк в таблице. Когда она считает значения столбца, она используется с DISTINCT, чтобы производить счёт чисел различных значений в данном поле.

COUNT(*) возвращает количество строк в таблице

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

COUNT(*)
5

COUNT(*выражение***)** возвращает количество строк с определенными значениями (не NULL) для выражения

Пример. Вывод количества сотрудников отдела 80, получающих комиссионные, из таблицы EMPLOYEES

```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;
```

COUNT(COMMISSION_PCT)
3

Использование ключевого слова DISTINCT

DISTINCT, сопровождаемый именем поля, с которым он применяется, помещён в круглые скобки, но не сразу после **SELECT**, как раньше. Такого использования **DISTINCT** с **COUNT**, применяемого к индивидуальным столбцам, требует стандарт ANSI, но большое количество программ не предъявляют такого требования.

COUNT(DISTINCT *выражение***)** возвращает количество уникальных определенных значений выражения.

DISTINCT может использоваться таким образом с любой агрегатной функцией, но наиболее часто он используется с **COUNT**. С **MAX** и **MIN** это просто не будет иметь никакого эффекта, а **SUM** и **AVG** вы обычно применяете для включения повторяемых значений, так как они эффективнее общих и средних значений всех столбцов.

COUNT со звёздочкой включает и **NULL**, и дубликаты; по этой причине **DISTINCT** не может быть использован. **DISTINCT** может производить более высокие числа, чем **COUNT** особого поля, который

удаляет все строки, имеющие избыточные или NULL-данные в этом поле. **DISTINCT** неприменим с **COUNT** (*), потому что он не имеет никакого действия в хорошо разработанной и поддерживаемой БД. В такой БД не должно быть ни таких строк, которые являлись бы полностью пустыми, ни дубликатов (первые не содержат никаких данных, а последние полностью избыточны). Если всё-таки имеются полностью пустые или избыточные строки, вы, вероятно, не захотите, чтобы **COUNT** скрыл от вас эту информацию.

Пример. Вывод количества уникальных номеров отделов из таблицы **EMPLOYEES**

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

COUNT(DISTINCTDEPARTMENT_ID)
7

Групповые функции и неопределенные значения

Групповые функции игнорируют неопределенные значения в столбцах

```
SELECT AVG(commission_pct)
FROM   employees;
```

AVG(COMMISSION_PCT)
.2125

Использование функции **NVL** с групповыми функциями

Функция **NVL** заставляет групповые функции включать неопределенные значения

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

AVG(NVL(COMMISSION_PCT,0))
.0425

Создание групп данных

До этого времени все групповые функции обрабатывали таблицу как одну большую группу информации. Сейчас необходимо разбить таблицу с информацией на меньшие группы. Это можно сделать, используя предложение **GROUP BY**.

Предложение **GROUP BY** позволяет вам определять подмножество значений в особом поле в терминах другого поля и применять агрегатную функцию к подмножеству. Это дает возможность объединять поля и агрегатные функции в едином предложении **SELECT**.

GROUP BY применяет агрегатные функции, независимо от серий групп, которые определяются с помощью значения поля в целом. В этом случае каждая группа состоит из всех строк с тем же самым значением поля snum, и **MAX** функция применяется отдельно для каждой такой группы. Это значение поля, к которому применяется **GROUP BY**, имеет, по определению, только одно значение на группу вывода так же, как это делает агрегатная функция. Результатом является совместимость, которая позволяет агрегатам и полям объединяться таким образом.

Использование предложения GROUP BY

```
SELECT        [столбец,] групп_функция(столбец), ...
FROM          таблица
[WHERE        условие]
[GROUP BY     выражение_группировки]
[ORDER BY     столбец];
```

Предложение **GROUP BY** разбивает строки таблицы на группы.

Все столбцы, которые входят в список **SELECT** и к которым не применяются групповые функции, должны быть указаны в предложении **GROUP BY**.

```
SELECT department_id, AVG(salary)
FROM   employees
GROUP BY department_id;
```

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

8 rows selected.

Столбец, указанный в **GROUP BY**, может отсутствовать в списке **SELECT**.

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id;
```

AVG(SALARY)
4400
9500
3500
6400
10033.3333
19333.3333
10150
7000

8 rows selected.

В этом примере SQL выводит среднее количество продаж для каждого департамента, без отображения номера соответствующего департамента.

Группировка по нескольким столбцам

EMPLOYEES

DEPARTMENT_ID	JOB_ID	SALARY
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	3600
50	ST_CLERK	3100
50	ST_CLERK	2600
50	ST_CLERK	2500
50	ST_MAN	5800
60	IT_PROG	9000
60	IT_PROG	6000
60	IT_PROG	4200
80	SA_MAN	10500
80	SA_REP	11000
110	AC_MGR	12000
	SA_REP	7000

20 rows selected.

Просуммировать оклады в таблице EMPLOYEES по каждой должности внутри каждого отдела

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
	SA_REP	7000

13 rows selected.

Иногда возникает необходимость увидеть результаты для группы внутри другой группы. Пример показывает отчет, который выводит суммарное количество продаж для каждого типа работы, внутри каждого департамента. В предложении **GROUP BY**, через запятую, сначала указывается основная группа, а следом подгруппа предыдущей группы.

Использование предложения **GROUP BY** с несколькими столбцами

```
SELECT   department_id dept_id, job_id, SUM(salary)
FROM     employees
GROUP BY department_id, job_id;
```

DEPT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000

13 rows selected.

Недействительные запросы с групповыми функциями

Все столбцы и выражения из списка **SELECT**, не являющиеся групповой функцией, должны быть включены в предложение **GROUP BY**.

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

В предложении GROUP BY недостает столбца

```
SELECT department_id, COUNT(last_name)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

- Предложение **WHERE** для исключения групп не используется;
- Для исключения некоторых групп следует пользоваться предложением **HAVING;**
- Нельзя использовать групповые функции в предложении **WHERE.**

Предложение **HAVING** определяет критерии, используемые, чтобы удалять определенные группы из вывода, точно так же, как предложение **WHERE** делает это для отдельных строк.

Аргументы в предложении **HAVING** следуют тем же самым правилам, что и в предложении **SELECT**, состоящем из команд, использующих **GROUP BY**. Они должны иметь одно значение на группу вывода.

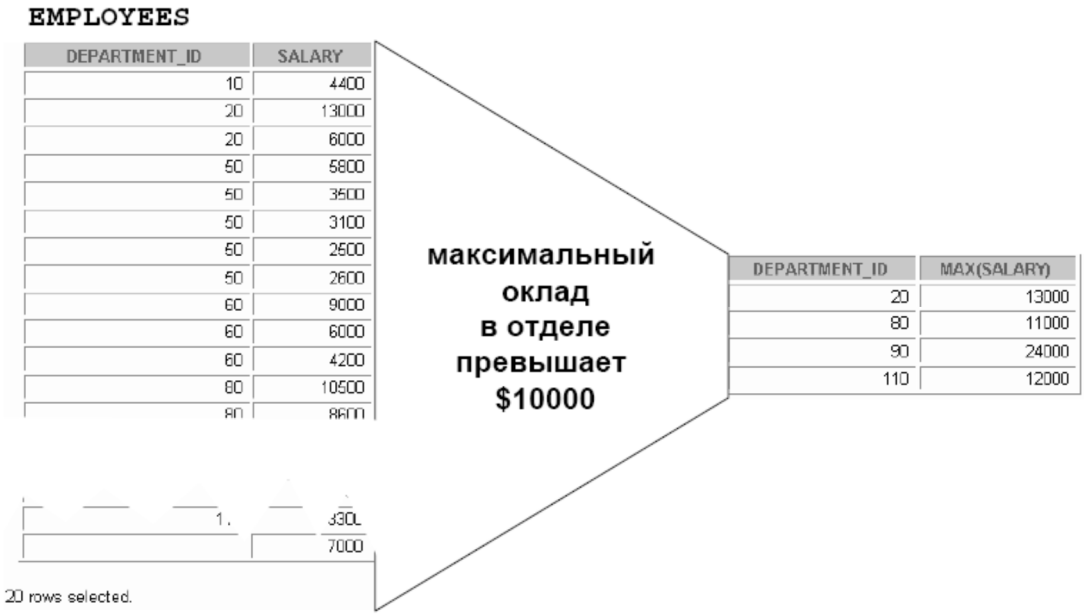
```
SELECT department_id, AVG(salary)
FROM   employees
WHERE  AVG(salary) > 8000
GROUP BY department_id;
```

Предложение WHERE для исключения групп не используется

```
WHERE  AVG(salary) > 8000

ERROR at line 3:
ORA-00934: group function is not allowed here
```

Исключение групп



Для исключения групп пользуйтесь предложением **HAVING**.

1. Строки группируются.
2. Применяется групповая функция.
3. Выводятся группы, удовлетворяющие условию в предложении **HAVING**.

```
SELECT        [столбец,] групп_функция(столбец), ...
FROM          таблица
[WHERE        условие]
[GROUP BY     выражение_группировки]
[HAVING       ограничивающее_условие]
[ORDER BY     столбец];
```

Примеры использования предложения HAVING

```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

JOB_ID	PAYROLL
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

Вложенные групповые функции

Вывод максимального среднего оклада

```
SELECT MAX(AVG(salary))
FROM   employees
GROUP BY department_id;
```

MAX(AVG(SALARY))
19333.3333

Выводы по теме №5

- Использование групповых функций COUNT, MAX, MIN, AVG.
- Запросы, использующие предложение GROUP BY.
- Запросы, использующие предложение HAVING.

Контрольные задания по теме №5

1. Групповые функции работаю с множеством строк для получения одного результата на каждую группу. Да/Нет?

2. Групповые функции включают null-значения в расчеты. Да/Нет?

3. Предложение WHERE выбирает строки до включения их в групповые расчеты. Да/Нет?

4. Отобразите минимальное, максимальное, суммарное и среднее значение продаж для всех сотрудников. Назовите столбцы Maximum, Minimum, Sum и Average, соответственно. Округлите результаты до ближайшего целого числа. Сохраните текст запроса в файл lab5_4.sql.

5. Отредактируйте запрос из файла lab5_4.sql так, что бы он отображал минимальное, максимальное, суммарное и среднее значение продаж для каждого типа работ (job_id). Сохраните текст запроса в файл lab5_5.sql.

6. Напишите запрос, отображающий количество людей с одинаковым типом работы.

7. Определите количество менеджеров, не перечисляя их. Назовите столбец Number of Managers. Совет: Для определения количества менеджеров используйте столбец MANAGER_ID.

8. Напишите запрос, отображающий разницу между максимальными и минимальными продажами. Назовите столбец DIFFERENCE.

9. Отобразите номера начальников (MANAGER_ID) и продажи сотрудника этого начальника, у которого наименьшие продажи. Исключите всех сотрудников, чей начальник неизвестен. Исключите группы, в которых минимальные продажи меньше $6000. Отсортируйте результат по убыванию продаж.

10. Напишите запрос, отображающий название всех департаментов, расположение (location_id), количество сотрудников и средние продажи всех сотрудников в данном департаменте. Назовите столбцы Name, Location, Number of People и Salary, соответственно. Округлите средние продажи до сотых.

11. Создайте запрос, который будет отображать общее количество сотрудников, и сколько сотрудников было принято на работу в 1995, 1996, 1997 и 1998 годах. Дайте столбцам соответствующие названия.

12. Создайте запрос для отображения в виде матицы типа работы, продаж для этого типа работы для данного номера департамента, и общее количество продаж для этого типа работы. Запрос создать для департаментов 20, 50, 80 и 90. Дайте каждому столбцу соответствующее заглавие. Ниже приведен пример результата, который необходимо получить:

JOB_ID	DEPT20	DEPT50	DEPT80	DEPT90	TOTAL
AC_ACCOUNT					8300
AC_MGR					12000
AD_ASST					4400
AD_PRES				24000	24000
AD_VP				34000	34000
FI_ACCOUNT					39600
FI_MGR					12000
HR_REP					6500
IT_PROG					28800
MK_MAN	13000				13000
MK_REP	6000				6000
PR_REP					10000
PU_CLERK					13900
PU_MAN					11000
SA_MAN			61000		61000
SA_REP			243500		250500
SH_CLERK		64300			64300
ST_CLERK		55700			55700
ST_MAN		36400			36400

19 rows selected.

ТЕМА 6. ПОДЗАПРОСЫ

Рассматриваемые вопросы:

- Типы проблем, решаемых с помощью подзапросов;
- Определение подзапросов;
- Типы подзапросов;
- Написание однострочных и многострочных подзапросов;
- Создание подзапросов для выборки данных по неизвестным критериям;
- Использование подзапросов для выявления значений, существующих в одном наборе данных и отсутствующих в другом.

С помощью SQL вы можете вкладывать запросы друга в друга. Обычно внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса, определяющего, верно оно или нет. Чтобы оценить внешний (основной) запрос, SQL сначала должен оценить внутренний запрос (или подзапрос) внутри предложения **WHERE**.

Использование подзапроса для решения проблемы

У кого оклад больше, чем у Абеля?

Предположим, что Вы хотите написать запрос для нахождения сотрудников, зарабатывающих больше чем Абель. Для решения этой проблемы необходимы два запроса: первый для нахождения зарплаты Абеля, и второй запрос для нахождения тех, у кого зарплата больше, чем найденное значение.

Вы можете решить эту проблему комбинированием двух запросов, разместив один ***внутри*** другого. Внутренний запрос, так же называемый ***подзапросом***, возвращает значение, которое используется внешним (основным) запросом.

Синтаксис подзапросов

```
SELECT    список_выбора
FROM      таблица
WHERE     выражение оператор
                    (SELECT       список_выбора
                    FROM          таблица);
```

- Подзапрос (внутренний запрос) выполняется один раз до главного запроса.
- Результат подзапроса используется главным запросом (внешним запросом).

Подзапрос это предложение **SELECT**, которое вставляется в предложения другого предложения **SELECT**. Вы можете создавать мощные запросы, основанные на множестве простых запросов, используя подзапросы. Это может быть очень полезным, когда необходимо выбрать строки из таблицы с условием, которое основано на данных из этой же таблицы.

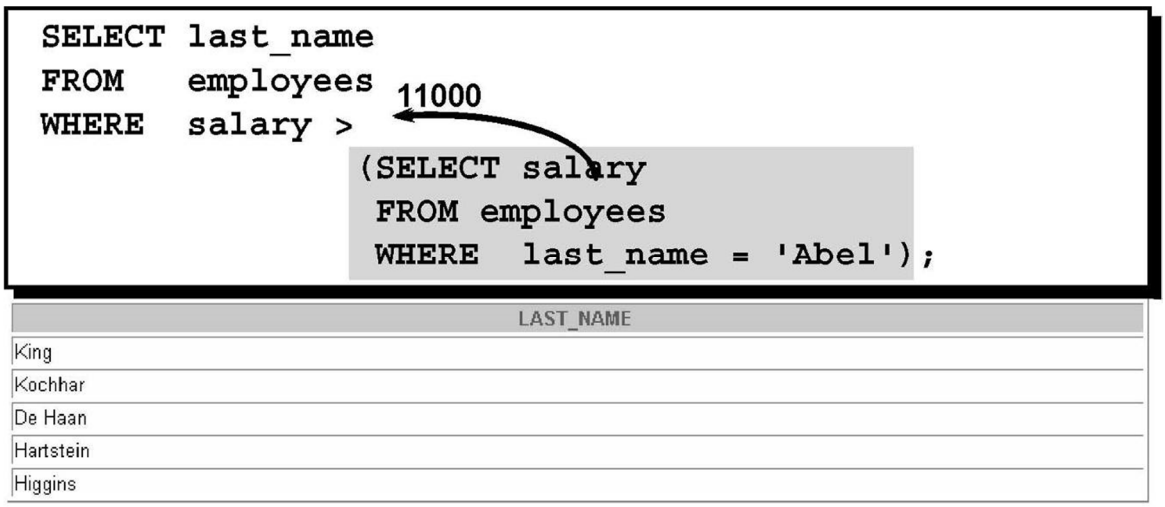
Вы можете размещать подзапросы в большом количестве SQL предложений, включая:

- **WHERE**;
- **HAVING**;
- **FROM**.

Подзапросы обычно относят к вложенным предложениям **SELECT**, под-SELECT-у или вложенным предложениям SELECT. Обычно подзапросы выполняются первыми, и их результат используется для завершения условия внешнего запроса.

При использовании подзапросов в предикатах, основанных на реляционных операциях, вы должны убедиться, что использовали подзапрос, который будет выдавать одну, и только одну, строку вывода. Если вы используете подзапрос, который не выводит никаких значений вообще, команда не потерпит неудачи, но основной запрос не выведет никаких значений. Подзапросы, которые не производят никакого вывода (или нулевой вывод), вынуждают рассматривать предикат ни как верный, ни как неверный, а как неизвестный. Однако неизвестный предикат имеет тот же самый эффект, что и неверный: никакие строки не выбираются основным.

Использование подзапроса



Данный запрос находит сотрудников, у которых оклад больше, чем у Абеля.

Указания по использованию подзапросов

- Подзапрос должен быть заключен в скобки.
- Подзапрос должен находиться справа от оператора сравнения.

- В подзапросе не следует использовать предложение **ORDER BY**, если только не применяется “TOP-N” анализ.
- В однострочных подзапросах используются однострочные операторы.
- В многострочных подзапросах используются многострочные операторы.

Типы подзапросов

Однострочный подзапрос

Многострочный подзапрос

Примечание: Так же существуют подзапросы, возвращающие более одного столбца (multiple-column subqueries).

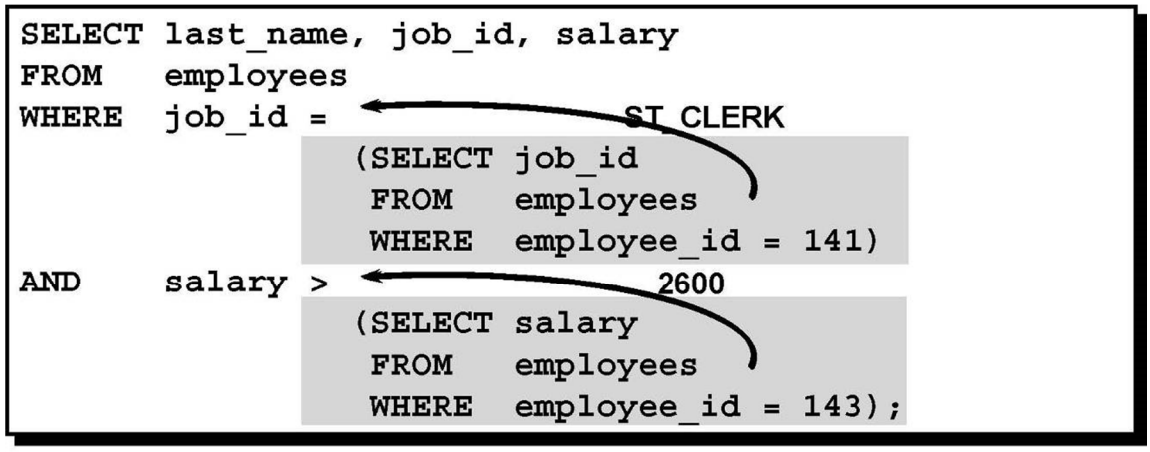
Однострочные подзапросы

- Возвращают только одну строку
- Используют однострочные операторы сравнения

Оператор	Значение
=	Равно
>	Больше чем

>=	Больше чем или равно
<	Меньше чем
<=	Меньше чем или равно
<>	Неравно

Выполнение однострочных подзапросов



LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

Этот пример показывает как выбрать сотрудников, которые занимают такую же должность как сотрудник номер 141 и которые зарабатывают больше, чем сотрудник номер 143. Пример содержит 3 блока запросов: внешний запрос и два внутренних запроса. Внутренние блоки запроса выполняются первыми, выдавая результаты ST_CLERK и 2600, соответственно. Затем обрабатывается внешний блок запроса, используя значения, возвращенные внутренними запросами, для завершения условия поиска.

Оба внутренних запроса возвращают по одному значению (ST_CLERK и 2600, соответственно), поэтому этот SQL запрос называется однострочный подзапрос.

Примечание: Внешний и внутренний запрос могут получать данные из разных таблиц.

Использование групповых функций в подзапросах

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =
                (SELECT MIN(salary)
                 FROM   employees);
```

2500

LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

Данный пример отображает фамилию сотрудника, тип работы, заработок всех сотрудников, чей заработок равен минимальному заработку. Групповая функция MIN возвращает единственное значение (2500) во внешний запрос.

Предложение HAVING с подзапросами

- Сервер Oracle сначала выполняет подзапрос
- Сервер Oracle возвращает результаты в предложение **HAVING** главного запроса

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
HAVING   MIN(salary) >
                        (SELECT MIN(salary)
                         FROM employees
                         WHERE department_id = 50);
```

2500

Подзапросы можно использовать не только в предложении **WHERE**, но также в предложении **HAVING**. Сервер Oracle исполняет подзапрос, и возвращает результат в предложение **HAVING** основного запроса. Вы можете также использовать подзапросы внутри предложения **HAVING**. Эти подзапросы могут использовать свои собственные агрегатные функции, если

они не производят нескольких значений, или использовать **GROUP BY** или **HAVING**.

Многострочные подзапросы

- Возвращают более одной строки
- Используют многострочные операторы сравнения

Оператор	Значение
IN	Равно любому члену списка
ANY	Сравнение значения с любым значением, возвращаемым подзапросом
ALL	Сравнение значения с каждым значением, возвращаемым подзапросом

Вы можете использовать подзапросы, которые производят любое число строк, если вы применяете специальный оператор **IN** (операторы **BETWEEN, LIKE и IS NULL** не могут использоваться с подзапросами). Как вы помните, **IN** определяет набор значений, одно из которых должно совпадать с другим термином уравнения предиката в заказе, чтобы предикат был верным.

Когда вы используете **IN** с подзапросом, SQL просто формирует этот набор из вывода подзапроса. Мы можем, следовательно, использовать **IN** чтобы выполнить такой подзапрос, который не будет работать с реляционным оператором, и найти все атрибуты таблицы

Использование оператора **ANY** в многострочных подзапросах

Операторы **SOME** и **ANY** взаимозаменяемы везде, и там, где мы используем **ANY, SOME** будет работать точно так же. Различие в терминологии состоит в том, чтобы позволить людям использовать тот термин, который является однозначным.

Оператор **ANY** берёт все значения, выведенные подзапросом, и оценивает их как верные, если любое (**ANY**) из них равняется значению

текущей строки внешнего запроса.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
206	Gietz	AC_ACCOUNT	8300

10 rows selected.

Оператор ANY может использовать другие реляционные операторы, помимо равно (=), и, таким образом, делать сравнения, которые превосходят возможности IN.

Использование оператора **ALL** в многострочных подзапросах

С помощью ALL, предикат будет верным, если каждое значение, выбранное подзапросом, удовлетворяет условию в предикате внешнего запроса.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

ALL чаще используется с неравенствами, нежели с равенствами, так как значение может быть "равным для всех" результатом подзапроса, только если все результаты фактически идентичны.

Одно существенное различие между **ALL** и **ANY** - способ действия в ситуации, когда подзапрос не возвращает никаких значений. В принципе

всякий раз, когда допустимый подзапрос не в состоянии сделать вывод, **ALL** автоматически правилен, а **ANY** автоматически неправилен.

Неопределенные значения в подзапросе

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                              (SELECT mgr.manager_id
                               FROM   employees mgr);

no rows selected
```

Данный запрос должен отобразить всех сотрудников, у которых нет подчиненных. Если рассуждать логически, то запрос должен вернуть 12 строк. Однако запрос не возвращает строк. Одно из значений, которое возвращает внутренний запрос неопределенно (null), соответственно внешний запрос не возвращает строк. Причиной этого является то, что операторы, сравнивающие неопределенные значения (null) в результате дают неопределенность (null). Таким образом, когда результатом подзапроса может быть неопределенное значение, не используйте оператор NOT IN. Вместо оператора NOT IN используйте оператор <> ALL.

Когда SQL сравнивает два значения в предикате, одно из которых пустое (NULL), то результат неизвестен. Неизвестный предикат подобен неверному и является причиной того, что строка не выбирается, но работать он будет иначе в некоторых похожих запросах, в зависимости от того, используют они **ALL** или **ANY.**

Примечание: С оператором **IN** таких проблем не возникает.

Выводы по теме №6

Проблемы, решаемые с помощью подзапросов:

Создавайте подзапросы, когда запрос основан на неизвестных значениях;

```
SELECT    список_выбора
FROM      таблица
WHERE     выражение оператор
                    (SELECT       список_выбора
                       FROM       таблица);
```

Контрольные задания по теме №6

1. Напишите запрос, отображающий фамилию и дату приема на работу сотрудников, работающих в том же департаменте что и Zlotkey. Исключите из результата сотрудника Zlotkey.

2. Создайте запрос для отображения номера сотрудника, фамилию и заработную плату (salary) всех сотрудников, которые имеют зарплату больше средней. Отсортируйте результат по возрастанию заработной платы.

3. Напишите запрос, который отобразит номер сотрудника и фамилию всех сотрудников, которые работают в департаменте с сотрудником в фамилии которого есть буква “*u*”. Сохраните запрос в файл lab6_3.sql.

4. Отобразите фамилию, номер департамента и тип работы всех сотрудников в департамента с location_id (идентификатор местоположения) равным 1700.

5. Отобразите фамилию и заработную плату всех сотрудников, которые отчитываются King-у.

6. Отобразите номер департамента, фамилию и тип работы для сотрудников в департаменте “Executive”.

7. Отредактируйте запрос lab6_3.sql для отображения номера сотрудника, фамилии и заработной платы для всех сотрудников кто зарабатывает больше средней заработной платы и работает в департаменте с сотрудником, в чьей фамилии есть буква “*u*”. Сохраните запрос в файл lab6_7.sql.

ТЕМА 7. ФОРМИРОВАНИЕ И ВЫВОД ДАННЫХ С ПОМОЩЬЮ ISQL*PLUS

Рассматриваемые вопросы:

- Запросы, требующие входных переменных;
- Настройка среды iSQL*Plus;
- Форматирование выходных данных;
- Создание и выполнение скрипт-файлов;
- Создание запроса для вывода значений с помощью переменных подстановки;
- Запуск скрипт-файла, содержащего переменные подстановки.

Более детально с вопросами темы 7 в плане теории можно самостоятельно ознакомиться в источнике [7, глава 5].

Переменные подстановки

Использование переменных подстановки iSQL*Plus для временного хранения значений:

– одиночный амперсанд (&);
– двойной амперсанд (&&);
– команда **DEFINE**.

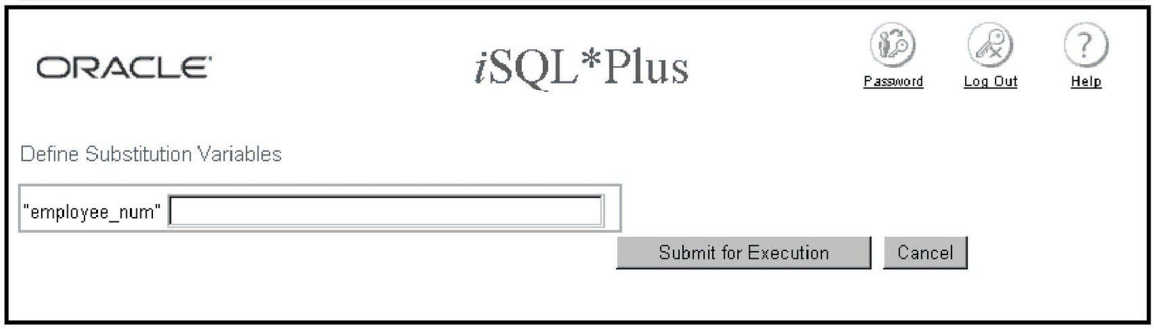
Передача значений переменных из одной команды SQL в другую.

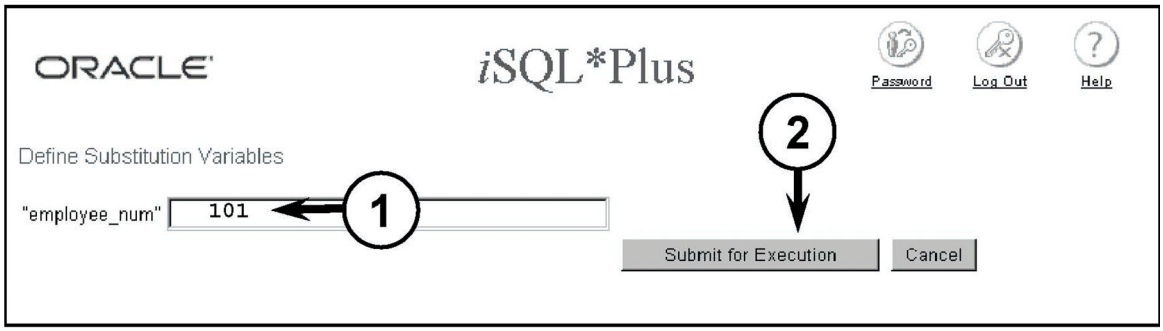
Динамическое изменение верхних и нижних колонтитулов.

Переменная подстановки с одним амперсандом (&)

Переменная с одним амперсандом (&) позволяет запросить значение у пользователя.

```
SELECT   employee_id, last_name, salary, department_id
FROM     employees
WHERE    employee_id = &employee_num ;
```





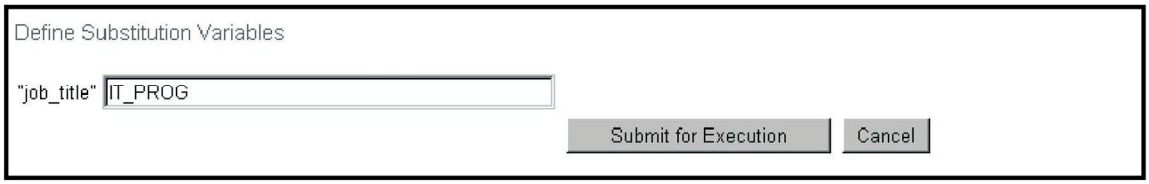

```
old 3: WHERE employee_id = &employee_num
new 3: WHERE employee_id = 101
```

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
101	Kochhar	17000	90

Символьные значения и даты с переменными подстановки

Даты и символьные значения заключаются в апострофы.

```
SELECT last_name, department_id, salary*12
FROM   employees
WHERE  job_id = '&job_title' ;
```



LAST_NAME	DEPARTMENT_ID	SALARY*12
Hunold	60	108000
Ernst	60	72000
Lorentz	60	50400

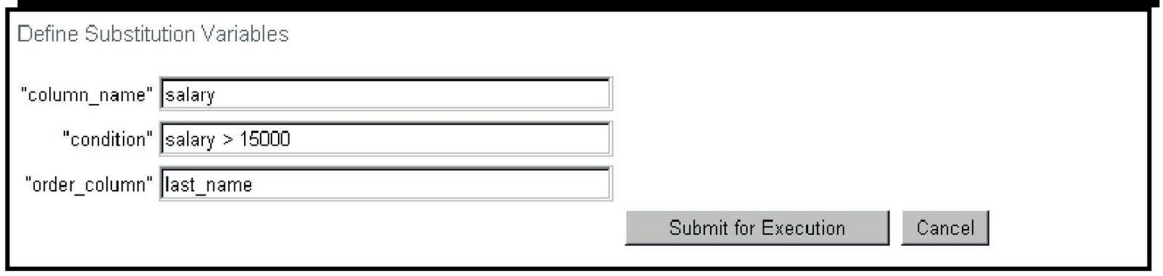
Примечание: Вы также можете использовать функции, например, UPPER и LOWER. Используйте UPPER('&job_title') что бы пользователю не надо было вводить job_id в верхнем регистре символов.

Задание имен столбцов, выражений и текста во время выполнения команды SQL

Переменные подстановки могут замещать:

- Условие WHERE;
- Предложение ORDER BY;
- Выражение столбца;
- Имя таблицы;
- Целую команду SELECT;

```
SELECT      employee_id, last_name, job_id,
            &column_name
FROM        employees
WHERE       &condition
ORDER BY    &order_column ;
```



EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
102	De Haan	AD_VP	17000
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000

Определение переменных подстановки

Задать переменную в iSQL*Plus можно с помощью команды DEFINE:

DEFINE переменная = значение

(создает пользовательскую переменную с типом данных CHAR).

Если в команде DEFINE требуется одиночный пробел, этот пробел должен быть заключен в апострофы.

Переменная остается заданной:

- до ее удаления командой UNDEFINE;
- до выхода из iSQL*Plus.

Проверить изменения можно с помощью команды DEFINE.

```
DEFINE job_title = IT_PROG
DEFINE job_title
DEFINE JOB_TITLE        = "IT_PROG" (CHAR)
```

```
UNDEFINE job_title
DEFINE job_title
SP2-0135: symbol job_title is UNDEFINED
```

Использование команды DEFINE с переменными подстановки

Создайте переменную подстановки, используя команду DEFINE.

```
DEFINE employee_num = 200
```

Используйте амперсанд (&) в качестве префикса переменной подстановки в команде SQL.

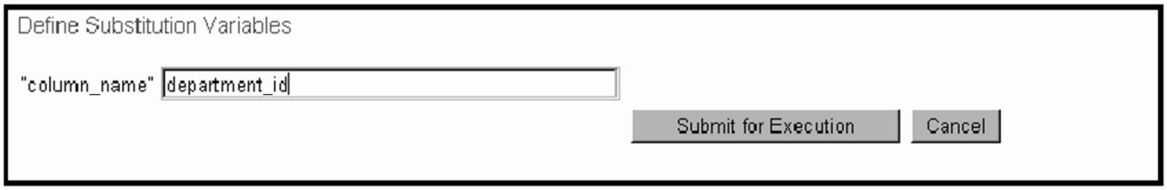
```
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num ;
```

EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
200	Whalen	4400	10

Переменные подстановки с двумя амперсандами (&&)

Переменная подстановки с двумя амперсандами (&&) позволяет многократно использовать значение переменной, не запрашивая его повторно у пользователя.

```
SELECT   employee_id, last_name, job_id, &&column_name
FROM     employees
ORDER BY &column_name;
```



EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
200	Whalen	AD_ASST	10
201	Hartstein	MK_MAN	20

...

20 rows selected.

Использование команды VERIFY

Если задан режим SET VERIFY ON, iSQL*Plus выводит текст команды до и после замены переменных подстановки значениями.

```
SET VERIFY ON
SELECT employee_id, last_name, salary, department_id
FROM   employees
WHERE  employee_id = &employee_num;
```


```
old   3: WHERE  employee_id = &employee_num
new   3: WHERE  employee_id = 200
```

Настройка среды iSQL*Plus

Для управления текущим сеансом пользуйтесь командой SET

```
SET system_variable value
```

Проверяйте заданные параметры с помощью команды SHOW

```
SET ECHO ON
```

```
SHOW ECHO
echo ON
```

Переменные команды SET

Переменные команды SET	Описание
FEED[BACK] {<u>6</u> | n | OFF | ON}	Показывает количество строк, возвращаемых запросом, если запрос выбрал хотя бы n строк
HEA[DING] {OFF | <u>ON</u>}	Определяет, будут ли выводиться заголовки столбцов
LONG {<u>80</u> | n}	Устанавливает максимальную ширину отображения значений типа LONG

Примечание: Значение n – это цифровое значение. Подчеркнутые значения указывают на значения по умолчанию. Если значение переменной не указывается, то iSQL*Plus устанавливает значение по умолчанию.

Команды форматирования среды iSQL*Plus

Команда	Описание
COL[UMN] [*column option*]	Устанавливает формат столбца
TTI[TLE] [text | OFF | ON]	Устанавливает заглавие вверху каждой

	страницы отчета
BTI[TLE] [text | OFF | ON]	Устанавливает заглавие внизу каждой страницы отчета
BRE[AK] [ON *report_element*]	Убирает повторяющиеся значения и разделяет строки на секции

Команда COLUMN

Управляет форматом вывода столбца:

```
COL[UMN] [{столбец|псевдоним} [опция]]
```

- CLE[AR]: сбрасывает все форматы столбца
- HEA[DING] текст: задает заголовок столбца
- FOR[MAT] формат: изменяет вывод столбца с помощью форматной модели
- NOPRINT | PRINT: спрятать столбец | показать столбец
- NULL текст: определяет текст выводимый вместо null-значений

Использование команды COLUMN

Создайте заголовки столбцов

```
COLUMN last_name HEADING 'Employee|Name'
COLUMN salary JUSTIFY LEFT FORMAT $99,990.00
COLUMN manager FORMAT 999999999 NULL 'No manager'
```

Выведите на экран текущие установки для столбца LAST_NAME

```
COLUMN last_name
```

Сбросьте установки для столбца LAST_NAME.

```
COLUMN last_name CLEAR
```

Модели формата, используемые в команде COLUMN

Элемент	Описание	Пример	Результат
9	Один цифровой разряд	999999	1234
0	Ведущий ноль	099999	01234
$	Плавающий знак доллара	$9999	$1234

L	Местная валюта	L9999	$1234
.	Позиция десятичной точки	9999.99	1234.00
,	Разделитель тысяч	9,999	1,234

Создание скрипт-файла для выполнения отчета

1. Создайте и протестируйте команду SELECT языка SQL.
2. Сохраните команду SELECT в скрипт-файле.
3. Загрузите скрипт-файл в редактор.
4. Добавьте команды форматирования перед командой SELECT.
5. Убедитесь в том, что команда SELECT заканчивается символом завершения.
6. Установите значение по умолчанию для команд форматирования (сбросьте установки с использованием параметра CLEAR) после команды SELECT.
7. Сохраните скрипт-файл.
8. Загрузите скрипт-файл в окно текста iSQL*Plus и щелкните на кнопке Execute.

Образец отчета

```
SET FEEDBACK OFF
TTITE 'Employee|Report'
BTITLE 'Confidential'
BREAK ON job_id
COLUMN job_id HEADING 'Job|Category'
COLUMN last_name HEADING 'Employee'
COLUMN salary HEADING 'Salary' FORMAT $99,999.99
REM ** Insert SELECT statement
SELECT job_id, last_name, salary
FROM employees
WHERE salary < 15000
ORDER BY job_id, last_name
```

```
/
REM clear all formatting commands ...
SET FEEDBACK ON
COLUMN job_id CLEAR
COLUMN last_name CLEAR
COLUMN salary CLEAR
CLEAR BREAKE
```

Fri Sep 28 Employee Report page 1

Job Category	Employee	Salary
AC_ACCOUNT	Gietz	$8,300.00
AC_MGR	Higgins	$12,000.00
AD_ASST	Whalen	$4,400.00
IT_PROG	Ernst	$6,000.00
	Hunold	$9,000.00
	Lorentz	$4,200.00
MK_MAN	Hartstein	$13,000.00
MK_REP	Fay	$6,000.00
SA_MAN	Zlotkey	$10,500.00
SA_REP	Abel	$11,000.00
	Grant	$7,000.00
	Taylor	$8,600.00

Confidential

Выводы по теме №7

- Для временного хранения значений используются переменные подстановки iSQL*Plus.
- Для управления текущей средой iSQL*Plus используется команда SET.
- Для управления выводом столбцов используется команда COLUMN.
- Для удаления дубликатов и строк, по которым производится разбивка отчета, используется команда BREAK.
- Для вывода заголовков и нижних колонтитулов используются команды TTITLE и BTITLE.

Контрольные задания по теме №7

Следующий синтаксис верен:

DEFINE & p_val = 100

Да/Нет?

Команда DEFINE – это SQL команда. Да/Нет?

1. Напишите скрипт-файл для отображения фамилии, job_id и даты приема на работу для всех сотрудников, принятых на работу в заданном диапазоне дат. Объедините фамилию и job_id, разделяя их запятой и пробелом, и назовите столбец Employees. Используйте команду DEFINE для задания диапазона дат. Используйте формат MM/DD/YYYY. Сохраните скрипт в файл lab7_3.sql.

2. Напишите скрипт для отображения фамилии, job_id и названия департамента для каждого сотрудника с заданным местоположением департамента (location). Строка поиска должна позволять нечувствительный к регистру символов поиск местонахождения департамента. Сохраните скрипт в файл lab7_4.sql.

3. Измените lab7_4.sql для создания отчета содержащего название департамента, фамилию сотрудника, дату приема на работу, заработную плату и годовую заработную плату для каждого сотрудника с заданным местоположением департамента (location). Назовите столбцы DEPARTMENT NAME, EMPLOYEE NAME, START DATE, SALARY и ANNUAL SALARY, размещая название на нескольких строках. Сохраните скрипт в файл lab7_5.sql.

ТЕМА 8. СКАНИРОВАНИЕ УЯЗВИМОСТЕЙ СУБД ORACLE

Рассматриваемые вопросы:

- Основные методы и средства сканирования уязвимостей СУБД Oracle.
- Выполнение сканирование уязвимостей СУБД Oracle с помощью сканера безопасности XSpider.

В сканере безопасности xSpider СУБД рассматривается как сетевая служба, в отношении которой выполняется анализ заголовков. В СУБД Oracle включена сетевая служба, которая обеспечивает ее сетевую поддержку и всегда запущена – TNS Listener. Эта служба как отдельный процесс принимает клиентские запросы, которые передаются для обработки соответственному серверному процессу СУБД.

Сетевой сканер через службу TNS Listener осуществляет проверку следующих параметров защиты:

- локальная аутентификация на уровне установленной операционной системы;
- защита паролем;
- опция ADMIN_RESTRICTIONS.

Локальная аутентификация на уровне операционной системы включается путем добавления параметра LOCAL_OS_AUTHENTICATION в файл listener.ora. Если значение этого параметра установлено в положение ON, управлять сервисом Listener можно только локально, с консоли сервера. Начиная с версии ORACLE 10g R1 локальная аутентификация включена по умолчанию. Если же этот параметр выключен – управление сервером возможно удаленно. В общем случае удаленное управление позволяет выполнить следующие действия:

- получить детальную информацию о системе с помощью команды status;

- остановить сетевую службу TNS Listener с помощью команды STOP;

- внести изменения в систему с помощью команды SET.

Опция ADMIN_RESTRICTIONS запрещает удаленное выполнение команды SET, а защита паролем ограничивает получение детальной информации о системе и выполнение команд.

Выводы по теме №8

- В СУБД Oracle включена сетевая служба, которая обеспечивает ее сетевую поддержку и всегда запущена – TNS Listener.
- Сетевой сканер через службу TNS Listener позволяет проводить проверку ряд параметров защиты.

Контрольные задания по теме №8

1. Создать профиль сканирования “СУБД Oracle”. Список сканируемых портов ограничить значением 1342. Отключить подбор учетных записей и сканирование служб протокола транспортного уровня UDP.

2. Запустить сканирование сервера в рамках созданного профиля. Обратить внимание на статус проверки “Локальная аутентификация на уровне ОС”. Объяснить невозможность удаленного сбора информации о службах СУБД Oracle.

3. На запущенном сервере открыть для редактирования файл listener.ora. Присвоить параметру LOCAL_OS_AUTHENTICATION значение OFF. Перезапустить сетевую службу TNS Listener. Запустить повторное сканирование сервера. Проанализировать результаты. Обратить внимание на то, что локальная аутентификация отключена.

4. Установить ПО Oracle Client. Выполнить перезагрузку. В профиле сканирования включить подбор учетных записей. Проверить, что включены опции “Подбирать учетные записи для БД Oracle”, “Подбирать

пароли для указанных” и указать имя экземпляра testdb. Выполнить повторное сканирование сервера.

ТЕМА 9. РЕЗЕРВНЫЙ СЕРВЕР И ЗЕРКАЛИРОВАНИЕ

Рассматриваемые вопросы:

- Особенности обеспечения доступности системы баз данных.
- Анализ и настройка технологии высокого уровня доступности и восстановления в аварийных ситуациях.

Для обеспечения доступности системы БД применяют несколько методов, к которым, в частности, относится: использование резервного сервера; использование технологии RAID. В основе указанных подходов рассматривают систему управления базами данных (*серверная система*), на основе которой выделяют следующие технологии: отказоустойчивая кластеризация; зеркальное отображение базы данных; пересылка журналов транзакций; высокий уровень доступности и восстановления в аварийных ситуациях (HARD).

Резервный сервер (Standby Server) — сервер, находящийся в резерве, на случай если что-либо случится с рабочим производственным сервером (также называемым первичным сервером) телекоммуникационной сети. Файлы, базы данных (системные и пользовательские) и учетные записи пользователей на резервном сервере идентичны этим элементам на производственном сервере.

Реализация резервного сервера осуществляется сначала восстанавливая на нем полную резервную копию базы данных, а затем последовательно восстанавливая резервные копии журнала транзакций производственного сервера, чтобы содержать всю информацию резервного сервера синхронизированной с информацией основного производственного сервера. Для установки резервного сервера необходимо присвоить опции баз данных “read only” значение TRUE. Эта опция запрещает пользователям любые операции записи в базу данных.

Общая процедура установки копии производственной базы данных на резервном сервере состоит из следующих этапов:

- восстанавливается рабочая база данных, используя инструкцию RESTORE DATABASE с предложением STANDBY;
- восстанавливаются все, кроме последней, резервные копии журнала транзакций, используя инструкцию RESTORE LOG с предложением STANDBY;
- восстанавливается последняя резервная копия журнала транзакций, используя инструкцию RESTORE LOG с предложением RECOVERY.

Последняя операция воссоздает БД, не создавая при этом файла с прообразами, делая БД доступной также и для операций записи. После восстановления БД и журналов транзакций пользователи получают для работы точную копию производственной БД. Потеряны будут только транзакции, которые не были зафиксированы при сбое.

Массив дисков RAID (Redundant Array of Independent Disks, избыточный массив независимых жестких дисков) представляет собой специальную конфигурацию дисков, в которой несколько приводов дисков составляют единую логическую единицу. Этот способ позволяет расположить каждый файл на нескольких физически разных дисках. Иными словами, хотя части файла находятся на разных дисках, система видит этот файл как единое целое.

Технология RAID позволяет повысить надежность за счет понижения производительности. Существует шесть основных уровней RAID, от 0 по 5. Только три из этих уровней представляют важность для систем баз данных: уровни 0, 1 и 5. Система RAID может быть реализована аппаратным или программным образом. Программная система RAID обычно поддерживается операционной системой. Операционная система Windows 7(8) поддерживает уровни RAID 0, 1 и 5. Технология RAID предоставляет защиту от сбоев жестких дисков и сопутствующей этим сбоям потери данных тремя

способами: чередованием дисков, зеркалированием дисков и контролем по четности.

Преимуществом зеркалирования на основе программного решения является то, что оно позволяет зеркалировать разделы диска, тогда как аппаратные решения обычно зеркалируют весь диск. Для зеркального отображения базы данных используются два сервера, и база данных с одного из них будет зеркалироваться на другой. Первый из этих серверов называется главным (основным), а второй — зеркальным (дублирующим).

Зеркальное отображение базы данных разрешает выполнять непрерывную передачу журнала транзакций с главного сервера на зеркальный. Копия операций журнала транзакций записывается в журнал транзакций зеркальной базы данных, и в ней выполняются эти операции. Если главный сервер выходит из строя, то приложения могут переключиться на базу данных на зеркальном сервере, не ожидая завершения процесса ее восстановления. В отличие от отказоустойчивой кластеризации, зеркальный сервер полностью кэширован и готов к работе, будучи синхронизован с главным сервером. Можно реализовать до четырех зеркальных резервных наборов.

Для реализации зеркалирования используется опция MIRROR TO в инструкции BACKUP DATABASE или в инструкции BACKUP LOG.

При зеркальном отображении базы данных также применяется третий сервер, называющийся следящим сервером (Witness Server). Этот сервер определяет, который из первых двух серверов является главным, а который зеркальным. Этот сервер применяется только для поддержки автоматического перехода на другой ресурс в случае сбоя первого.

Чтобы разрешить автоматическое переключение, необходимо включить режим синхронной работы, присвоив опции SAFETY в инструкции ALTER DATABASE значение FULL.

Зеркальное отображение базы данных как технология достижения высокого уровня доступности имеет несколько недостатков, включая такие:

- запросы только для чтения не могут выполняться на зеркальной базе данных;

- эту технологию можно применять только на двух экземплярах SQL Server;

- эта технология зеркалирует только объекты внутри базы данных, но объекты такие, как регистрационные имена входа в систему не могут быть защищены с использованием зеркалирования.

С целью устранения этих недостатков зеркалирования базы данных, в SQL вводится новая технология, называемая высоким уровнем доступности и восстановления в аварийных ситуациях (HADR — high availability and disaster recovery). Эта технология позволяет довести до максимума доступность баз данных.

Для конфигурирования технологии HADR необходимо выполнить следующую последовательность шагов:

1. Установить экземпляры баз данных на обоих узлах.

2. Разрешить возможность HADR на обоих экземплярах. Выбрать последовательность команд SQL Server Configuration Manager | SQL Server Services, щелкнуть правой кнопкой требуемый экземпляр и в контекстном меню выбрать опцию Properties. В открывшемся окне свойств экземпляра выбрать вкладку SQL HADR и на ней установить флажок Enable SQL HADR Service.

3. В первичном экземпляре создать группу обеспечения доступности. В обозревателе объектов среды Management Studio развернуть папку "Management", щелкнуть правой кнопкой узел "Availability Groups" и в контекстном меню выбрать опцию New Availability Group.

4. Запустить синхронизацию данных, нажав на странице Results диалогового окна New Availability Group кнопку Start Data Synchronization.

5. Протестировать группы обеспечения доступности, создав таблицу на первичной реплике (посредством инструкции CREATE TABLE с предложением ON PRIMARY) и вставив в нее несколько строк.

6. Протестировать обработку отказа. Для этого развернуть последовательность узлов Management | Availability Groups | AVG | Availability Replicas, щелкнуть правой кнопкой вторичную реплику и выбрать Force Failover. После этого первичная и вторичная реплики должны поменяться ролями.

Выводы по теме №9

- Для обеспечения доступности системы баз данных применяют несколько методов, к которым относится: использование резервного сервера; использование технологии RAID. Реализация указанных методов основана на использовании технологий: отказоустойчивая кластеризация; зеркальное отображение базы данных; пересылка журналов транзакций; высокий уровень доступности и восстановления в аварийных ситуациях (HARD).
- Для установки копии производственной базы данных на резервном сервере необходимо выполнить ряд этапов.
- Для конфигурирования технологии HADR необходимо выполнить последовательность из шести шагов.

Контрольные задания по теме №9

1. Определить с какой целью в системах управления базами данных используется резервный сервер. В чем состоит общая процедура установки копии производственной базы данных на резервном сервере ?
2. Определить недостатки зеркального отображения базы данных.
3. Как осуществляется конфигурирование технологии HADR и в чем сложность настройки серверной системы ?

ПЕРЕЧЕНЬ ЛИТЕРАТУРЫ

1. Петкович Д. Microsoft SQL Server 2012. Руководство для начинающих / Д. Петкович. — СПб. : БХВ-Петербург, 2013. — 816 с.
2. Бейли Л. Изучаем SQL / Л. Бейли. — СПб. : Питер, 2012. — 573 с.
3. Грофф Дж. Р. SQL. Полное руководство / Дж. Р. Грофф, П. Н. Вайнберг, Э. Дж. Оппель. — М. : Вильямс, 2015. — 959 с.
4. Грабер М. SQL для простых смертных / М. Грабер. — М. : Лори, 2014. — 378 с.
5. Кригель А. SQL. Библия пользователя / А. Кригель, Б. Трухнов. — М. : Вильямс, 2010. — 752 с.
6. Скотт У. Oracle 9i. Программирование на языке PL/SQL / У. Скотт. — М. : Лори, 2004. — 528 с.
7. Маклафлин Б. PHP и MySQL. Исчерпывающее руководство / Б. Маклафлин. — СПб. : Питер, 2014. — 544 с.
8. Колегов Д. Н. Лабораторный практикум по основам построения защищенных компьютерных сетей / Д. Н. Колегов. — Томск : Томский государственный университет, 2013. — 140 с.